Foundations and Trends® in Signal Processing

Bilevel Methods for Image Reconstruction

Suggested Citation: Caroline Crockett and Jeffrey A. Fessler (2021), "Bilevel Methods for Image Reconstruction", Foundations and Trends® in Signal Processing: Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

Caroline Crockett
University of Michigan
cecroc@umich.com

Jeffrey A. Fessler
University of Michigan
fessler@umich.edu

now
the essence of knowledge
Boston — Delft

Contents

1 Introduction **3**
- 1.1 Notation . . . 5
- 1.2 Defining a Bilevel Problem . . . 7
- 1.3 Running Example . . . 12
- 1.4 Conclusion . . . 15

2 Background: Cost Functions and Image Reconstruction **17**
- 2.1 Image Reconstruction . . . 17
- 2.2 Sparsity-Based Regularizers . . . 20
- 2.3 Brief History of Analysis Regularizer Learning . . . 25
- 2.4 Summary . . . 30

3 Background: Loss Functions and Hyperparameter Optimization **31**
- 3.1 Image Quality Metrics . . . 33
- 3.2 Parameter Search Strategies . . . 38
- 3.3 Summary . . . 43

4 Gradient Based Bilevel Methodology: The Groundwork **45**
- 4.1 Set-up . . . 46
- 4.2 Minimizer Approach . . . 46

4.3 Translation to a Single Level . . . 53
4.4 Unrolled Approaches . . . 56
4.5 Summary . . . 64

5 Gradient-Based Bilevel Optimization Methods 68
5.1 Double-Loop Algorithms . . . 69
5.2 Single-Loop Algorithms . . . 76
5.3 Complexity Analysis . . . 78
5.4 Summary of Methods . . . 87

6 Survey of Applications 91
6.1 Lower-level Cost Function Design . . . 92
6.2 Upper-Level Loss Function Design . . . 97
6.3 Conclusion . . . 101

7 Connections and Future Directions 103
7.1 Connection: Learnable Optimization Algorithms . . . 105
7.2 Connection: Equilibrium-based Networks . . . 107
7.3 Connection: Plug-and-play Priors . . . 110
7.4 Connection: Single-Level Parameter Learning . . . 112
7.5 Future Directions . . . 115
7.6 Summary of Advantages and Disadvantages . . . 118

Acknowledgements 122

Appendices 123

A Background: Primal-Dual Formulations 124

B Forward and Reverse Approaches to Unrolling 129

C Additional Running Example Results 134
C.1 Derivatives for Convolutional Filters . . . 134
C.2 Evaluating Assumptions for the Running Example . . . 136

D Implementation Details **142**
D.1 Vertical Bar Training Image 143
D.2 Cameraman Training Image 144

References **146**

Bilevel Methods for Image Reconstruction

Caroline Crockett[1] and Jeffrey A. Fessler[1]

[1] *Department of EECS, University of Michigan, Ann Arbor, Michigan, USA; {cecroc,fessler}@umich.edu*

ABSTRACT

This review discusses methods for learning parameters for image reconstruction problems using bilevel formulations. Image reconstruction typically involves optimizing a cost function to recover a vector of unknown variables that agrees with collected measurements and prior assumptions. State-of-the-art image reconstruction methods learn these prior assumptions from training data using various machine learning techniques, such as bilevel methods.

One can view the bilevel problem as formalizing hyperparameter optimization, as bridging machine learning and cost function based optimization methods, or as a method to learn variables best suited to a specific task. More formally, bilevel problems attempt to minimize an upper-level loss function, where variables in the upper-level loss function are themselves minimizers of a lower-level cost function.

This review contains a running example problem of learning tuning parameters and the coefficients for sparsifying filters used in a regularizer. Such filters generalize the popular total variation regularization method, and learned filters are closely related to convolutional neural networks approaches that are rapidly gaining in popularity. Here, the lower-level

Caroline Crockett and Jeffrey A. Fessler (2021), "Bilevel Methods for Image Reconstruction", Foundations and Trends® in Signal Processing: Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

problem is to reconstruct an image using a regularizer with learned sparsifying filters; the corresponding upper-level optimization problem involves a measure of reconstructed image quality based on training data.

This review discusses multiple perspectives to motivate the use of bilevel methods and to make them more easily accessible to different audiences. We then turn to ways to optimize the bilevel problem, providing pros and cons of the variety of proposed approaches. Finally we overview bilevel applications in image reconstruction.

1

Introduction

Methods for image recovery aim to estimate a good-quality image from noisy, incomplete, or indirect measurements. Such methods are also known as computational imaging. For example, image denoising and image deconvolution attempt to recover a clean image from a noisy and/or blurry input image, and image inpainting tries to complete missing measurements from an image. Medical image reconstruction aims to recover images that humans can interpret from the indirect measurements recorded by a system like a Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) scanner. Such image reconstruction applications are a type of inverse problem [1].

New methods for image reconstruction attempt to lower complexity, decrease data requirements, or improve image quality for a given input data quality. For example, in CT one goal is to provide doctors with information to help their patients while reducing radiation exposure [2]. To achieve these lower radiation doses, the CT system must collect data with lower beam intensity or fewer views. Similarly, in MRI collecting fewer k-space samples can reduce scan times. Such "undersampling" leads to an under-determined problem, with fewer knowns (measurements from a scanner) than unknowns (pixels in the reconstructed

image), requiring advanced image reconstruction methods.

Existing reconstruction methods make different assumptions about the characteristics of the images being recovered. Historically, the assumptions are based on easily observed (or assumed) characteristics of the desired output image, such as a tendency to have smooth regions with few edges or to have some form of sparsity [3]. More recent machine learning approaches use training data to discover image characteristics. These learning-based methods often outperform traditional methods, and are gaining popularity in part because of increased availability of training data and computational resources [4], [5].

There are many design decisions in learning-based reconstruction methods. How many parameters should be learned? What makes a set of parameters "good?" How can one learn these good parameters? Using a bilevel methodology is one systematic way to address these questions.

Bilevel methods are so named because they involve two "levels" of optimization: an upper-level loss function that defines a goal or measure of goodness (equivalently, badness) for the learnable parameters and a lower-level cost function that uses the learnable parameters, typically as part of a regularizer. The main benefits of bilevel methods are learning task-based hyperparameters in a principled approach and connecting machine learning techniques with image reconstruction methods that are defined in terms of optimizing a cost function, often called model-based image reconstruction methods. Conversely, the main challenge with bilevel methods is the computational complexity. However, like with neural networks, that complexity is highest during the training process, whereas deployment has lower complexity because it uses only the lower-level problem.

The methods in this review are broadly applicable to bilevel problems, but we focus on formulations and applications where the lower-level problem is an image reconstruction cost function that uses regularization based on analysis sparsity. The application of bilevel methods to image reconstruction problems is relatively new, but there are a growing number of promising research efforts in this direction. We hope this review serves as a primer and unifying treatment for readers who may already be familiar with image reconstruction problems and traditional regularization approaches but who have not yet delved into bilevel

methods.

This review lies at the intersection of a specific machine learning method, bilevel, and a specific application, filter learning for image reconstruction. For overviews of machine learning in image reconstruction, see [5], [6]. For an overview of image reconstruction methods, including classical, variational, and learning-based methods, see [7]. Finally, for historical overviews of bilevel optimization and perspectives on its use in a wide variety of fields, see [8], [9]. Within the image recovery field, bilevel methods have also been used, *e.g.*, in learning synthesis dictionaries [10].

The structure of this review is as follows. The remainder of the introduction defines our notation and presents a running example bilevel problem. Section 2 provides background information on the lower-level image reconstruction cost function and analysis regularizers. Section 3 provides background information on the upper-level loss function, specifically loss function design and hyperparameter optimization strategies. These background sections provide motivation and context for the rest of the review; they are not exhaustive overviews of these broad topics. Section 4 presents building blocks for optimizing a bilevel problem. Section 5 uses these building blocks to discuss optimization methods for the upper-level loss function. Section 6 discusses previous applications of the bilevel method in image recovery problems, including signal denoising, image inpainting, and medical image reconstruction. It also overviews bilevel formulations for blind learning and learning space-varying tuning parameters. Finally, Section 7 offers summarizing commentary on the benefits and drawbacks of bilevel methods for computational imaging, connects and compares bilevel methods to other machine learning approaches, and proposes future directions for the field.

1.1 Notation

This review focuses on continuous-valued, discrete space signals. Some papers, *e.g.*, [11], [12], analyze signals in function space, arguing that the goal of high resolution imagery is to approximate a continuous space reality and that analysis in the continuous domain can yield insights

and optimization algorithms that are resolution independent. However, the majority of bilevel methods are motivated and described in discrete space. The review does not include discrete-valued settings, such as image segmentation; those problems often require different techniques to optimize the lower-level cost function, although some recent work uses dual formulations to bridge this gap [13], [14].

The literature is inconsistent in how it refers to variables in machine learning problems. For consistency within this document, we define the following terms:

- **Hyperparameters**: Any adjustable parameters that are part of a model. Tuning parameters and model parameters are both sub-types of hyperparameters. This document uses $\boldsymbol{\gamma}$ to denote a vector of hyperparameters.
- **Tuning parameters**: Scalar parameters that weight terms in a cost function to determine the relative importance of each term. This review uses β to denote individual tuning parameters.
- **Model parameters**: Parameters, generally in vector or matrix form, that are used in the structure of a cost or loss function, typically as part of the regularization term. In the running example in the next section, the model parameters are typically filter coefficients, denoted $\boldsymbol{c}$.

We write vectors as column vectors and use bold to denote matrices (uppercase letters) and vectors (lowercase letters). Subscripts index vector elements, so x_i is the ith element in $\boldsymbol{x}$. For functions that are applied element-wise to vectors, we use notation following the Julia programming language [15], where $f.(\boldsymbol{x})$ denotes the function f applied element wise to its argument:

$$\boldsymbol{x} \in \mathbb{F}^N \implies f.(\boldsymbol{x}) = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix} \in \mathbb{F}^N.$$

We will often use this notation in combination with a transposed vector of ones to sum the result of a function applied element-wise to a vector, *i.e.*,

$$\mathbf{1}' f.(\boldsymbol{x}) = \sum_{i=1}^{N} f(x_i). \tag{1.1}$$

For example, the standard Euclidean norm is equivalent to $\mathbf{1}'f.(\boldsymbol{x})$ when $f(x) = |x|^2$ and and the vector 1-norm can be similarly written when $f(x) = |x|$. This notation is helpful for regularizers that do not correspond to norms. The field $\mathbb{F}$ can be either $\mathbb{R}$ or $\mathbb{C}$, depending on the application.

Convolution between a vector, $\boldsymbol{x}$, and a filter, $\boldsymbol{c}$, is denoted as $\boldsymbol{c} \circledast \boldsymbol{x}$. This review assumes all convolutions use circular boundary conditions. Thus, convolution is equivalent to multiplication with a square, circulant matrix:

$$\boldsymbol{c} \circledast \boldsymbol{x} = \boldsymbol{C}\boldsymbol{x}.$$

The conjugate mirror reversal of $\boldsymbol{c}$ is denoted as $\tilde{\boldsymbol{c}}$ and its application is equivalent to multiplying with the adjoint of $\boldsymbol{C}$:

$$\tilde{\boldsymbol{c}} \circledast \boldsymbol{x} = \boldsymbol{C}'\boldsymbol{x},$$

where the prime indicates the Hermitian transpose operation.

Finally, for partial derivatives, we use the notation that

$$\begin{aligned} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{y}) &= \frac{\partial f(\boldsymbol{x}, \boldsymbol{y})}{\partial \boldsymbol{x}} \in \mathbb{F}^N, \\ \nabla_{\boldsymbol{xy}} f(\boldsymbol{x}, \boldsymbol{y}) &= \left[\frac{\partial^2 f(\boldsymbol{x}, \boldsymbol{y})}{\partial x_i \partial y_j} \right] \in \mathbb{F}^{N \times M}, \text{ and} \\ \nabla_{\boldsymbol{xy}} f(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) &= \nabla_{\boldsymbol{xy}} f(\boldsymbol{x}, \boldsymbol{y}) \Big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}, \boldsymbol{y}=\hat{\boldsymbol{y}}} \in \mathbb{F}, \end{aligned} \tag{1.2}$$

where $f : \mathbb{F}^N \times \mathbb{F}^M \to \mathbb{F}$.

Tables 1.1 and 1.2 summarize our frequently used notation for variables and functions.

1.2 Defining a Bilevel Problem

This section introduces a generic bilevel problem; the next presents a specific bilevel problem that serves as a running example throughout the review. Later sections discuss many of the ideas presented here more thoroughly. Our hope is that an early introduction to the formal problem motivates readers and that this section acts as a quick-reference guide to our notation.

Variable	Dim	Description
$\boldsymbol{x}_j^{\text{true}}$	N	One of J clean, noiseless training signals. Often used in a supervised training set-up.
$\boldsymbol{A}$	$M \times N$	Forward operator for the system of interest.
$\boldsymbol{y}_j$	M	During the bilevel learning process, $\boldsymbol{y}_j$ refers to simulated measurements, where $\boldsymbol{y}_j = \boldsymbol{A}\boldsymbol{x}_j^{\text{true}} + \boldsymbol{n}_j$. Once $\boldsymbol{\gamma}$ is learned, $\boldsymbol{y}$ refers to collected measurements.
$\boldsymbol{n}_j$	N	A noise realization.
$\hat{\boldsymbol{x}}_j$	N	A reconstructed image.
$\boldsymbol{\gamma}$	R	The vector of parameters to learn using bilevel methods. This often includes $\boldsymbol{c}_k$ and/or β_k.
$\boldsymbol{c}_k$	S	One of K convolutional filters. A 2D filter might be $\sqrt{S} \times \sqrt{S}$.
$\tilde{\boldsymbol{c}}_k$	S	Conjugate mirror reversal of filter $\boldsymbol{c}_k$.
$\boldsymbol{C}_k$	$N \times N$	The convolution matrix such that $\boldsymbol{C}_k\boldsymbol{x} = \boldsymbol{c}_k \circledast \boldsymbol{x}$ and $\boldsymbol{C}_k'\boldsymbol{x} = \tilde{\boldsymbol{c}}_k \circledast \boldsymbol{x}$.
β_k	$\mathbb{R}$	The tuning parameter associated with $\boldsymbol{c}_k$.
β_0	$\mathbb{R}$	An overall regularization (tuning) parameter, appearing as e^{β_0} in (Ex).
$\boldsymbol{\Omega}$	$F \times N$	A matrix with filters in each row. For the stacked convolution matrices in (2.7) $F = KN$.
$\boldsymbol{z}$	Varies	A sparse vector, often from $\boldsymbol{C}_k\boldsymbol{x}$.
ϵ	$\mathbb{R}_+$	Parameter used to define ϕ. Typically determines the amount of corner-rounding.
t	$0, \ldots, T$	Iteration counter for the lower-level optimization iterates, *e.g.*, $\boldsymbol{x}^{(t)}$ is the estimate of the lower-level optimization variable $\boldsymbol{x}$ at the tth iteration.
u	$0, \ldots, U$	Iteration counter for the upper-level optimization iterates, *e.g.*, $\boldsymbol{\gamma}^{(u)}$.

Table 1.1: Overview of frequently used symbols in the review.

Function	Description
$\ell(\boldsymbol{\gamma}) \mapsto \mathbb{R}$ or $\ell(\boldsymbol{\gamma}, \boldsymbol{x}) \mapsto \mathbb{R}$	Upper-level loss function used as a fitness measure of $\boldsymbol{\gamma}$. Although ℓ is a function of $\boldsymbol{\gamma}$, it is often helpful to write it with two inputs, where typically $\boldsymbol{x} = \hat{\boldsymbol{x}}$.
$\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}) \mapsto \mathbb{R}$	Lower-level cost function used for reconstructing an image.
$R(\boldsymbol{x}) \mapsto \mathbb{R}$	Regularization function. Incorporates prior information about likely image characteristics.
$d(\boldsymbol{x}, \boldsymbol{y}) \mapsto \mathbb{R}$	Data-fit term.
$\phi(z) \mapsto \mathbb{R}$	Sparsity promoting function, *e.g.*, 0-norm, 1-norm, or corner-rounded 1-norm. Typically used in R.

Table 1.2: Overview of frequently used functions in the review.

This review considers the image reconstruction problem where the goal is to form an estimate $\hat{\boldsymbol{x}} \in \mathbb{F}^N$ of a (vectorized) latent image, given a set of measurements $\boldsymbol{y} \in \mathbb{F}^M$. For denoising problems, $N = M$, but the two dimensions may differ significantly in more general image reconstruction problems. The forward operator, $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ models the physics of the system such that one would expect $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ in an ideal (noiseless) system. We focus on linear imaging systems here, but the concepts generalize readily to nonlinear forward models. When known (in a supervised training setting), we denote the true, underlying signal as $\boldsymbol{x}^{\text{true}} \in \mathbb{F}^N$. Most bilevel methods are supervised, but Section 6.2 presents a few examples of unsupervised bilevel methods.

We focus on model-based image reconstruction methods where the goal is to estimate $\boldsymbol{x}$ from $\boldsymbol{y}$ by solving an optimization problem of the form

$$\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \underset{\boldsymbol{x} \in \mathbb{F}^N}{\operatorname{argmin}} \, \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}, \boldsymbol{y}). \tag{1.3}$$

To simplify notation, we drop $\boldsymbol{y}$ from the list of Φ arguments except where needed for clarity. The quality of the estimate $\hat{\boldsymbol{x}}$ can depend greatly on the choice of the hyperparameters $\boldsymbol{\gamma}$. Historically there have been numerous approaches pursued for choosing $\boldsymbol{\gamma}$, such as cross validation [16], generalized cross validation [17], the discrepancy principle [18] and Bayesian methods [19], among others.

Bilevel methods provide a framework for choosing hyperparameters. A bilevel problem for learning hyperparameters $\boldsymbol{\gamma}$ has the following "double minimization" form:

$$\hat{\boldsymbol{\gamma}} = \operatorname*{argmin}_{\boldsymbol{\gamma} \in \mathbb{F}^R} \underbrace{\ell(\boldsymbol{\gamma}\,;\, \hat{\boldsymbol{x}}(\boldsymbol{\gamma}))}_{\ell(\boldsymbol{\gamma})} \text{ where} \tag{UL}$$

$$\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \operatorname*{argmin}_{\boldsymbol{x} \in \mathbb{F}^N} \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}). \tag{LL}$$

Fig. 1.1 depicts a generic bilevel problem for image reconstruction. The upper-level (UL) loss function, $\ell : \mathbb{R}^R \times \mathbb{F}^N \mapsto \mathbb{R}$, quantifies how (not) good is a vector $\boldsymbol{\gamma}$ of learnable parameters. The upper-level depends on the solution to the lower-level (LL) cost function, Φ, which depends on $\boldsymbol{\gamma}$. The upper-level can also be called the outer optimization, with the lower-level being the inner optimization. Another terminology is leader-follower, as the minimizer of the lower-level follows where the upper-level loss leads. We will also write the upper-level loss function with a single parameter as $\ell(\boldsymbol{\gamma}) := \ell(\boldsymbol{\gamma}\,;\, \hat{\boldsymbol{x}}(\boldsymbol{\gamma}))$.

We write the lower-level cost as an optimization problem with "argmin" and thus implicitly assume that Φ has unique minimizer, $\hat{\boldsymbol{x}}$. The lower-level is guaranteed to have a unique minimizer when Φ is a strictly convex function of $\boldsymbol{x}$. (See Section 4 for more discussion of this point). More generally, there may be a set of lower-level minimizers, each having some possibly distinct upper-level loss function value. For more discussion, [8] defines optimistic and pessimistic versions of the bilevel problem for the case of multiple lower-level solutions.

Bilevel methods typically use training data. Specifically, one often assumes that a given set of J good quality images $\boldsymbol{x}_1^{\text{true}}, \ldots, \boldsymbol{x}_J^{\text{true}} \in \mathbb{F}^N$ are representative of the images of interest in a given application. (For simplicity of notation we assume the training images have the same size, but they can have different sizes in practice.) We typically generate corresponding simulated measurements for each training image using the imaging system model:

$$\boldsymbol{y}_j = \boldsymbol{A}\boldsymbol{x}_j^{\text{true}} + \boldsymbol{n}_j, \quad j = 1, \ldots, J, \tag{1.4}$$

where $\boldsymbol{n}_j \in \mathbb{F}^M$ denotes an appropriate random noise realization[1]. In

[1] A more general system model allows the noise to depend on the data and system

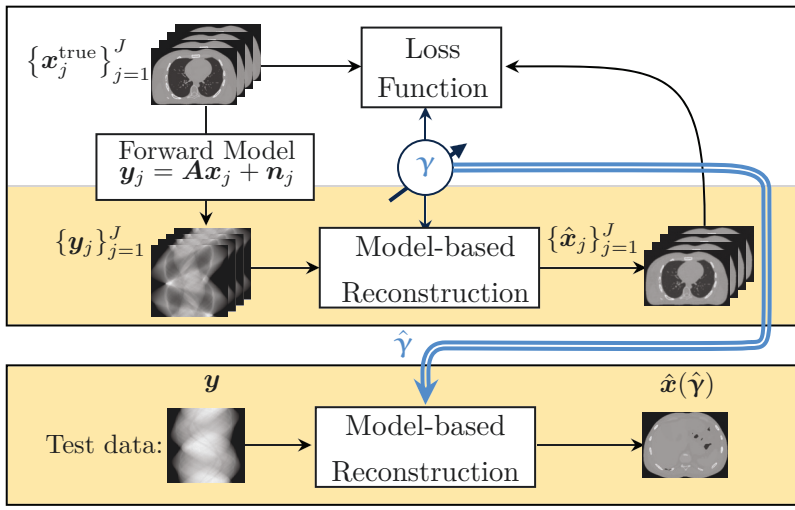


Figure 1.1: Depiction of a typical bilevel problem for image reconstruction, illustrated using XCAT phantom from [20]. The upper box represents the training process, with the upper-level loss and lower-level cost function. During training, one minimizes the upper-level loss with respect to a vector of parameters, $\boldsymbol{\gamma}$, that are used in the image reconstruction task. Once learned, $\hat{\boldsymbol{\gamma}}$ is typically deployed in the same image reconstruction task, shown in the lower box.

(1.4), we add one noise realization to each of the J images; in practice one could add multiple noise realizations to each $\boldsymbol{x}_j^{\text{true}}$ to augment the training data. We then use the training pairs $(\boldsymbol{x}_j^{\text{true}}, \boldsymbol{y}_j)$ to learn a good value of $\boldsymbol{\gamma}$. After those parameters are learned, we reconstruct subsequent test images using (1.3) with the learned hyperparameters $\hat{\boldsymbol{\gamma}}$.

An alternative to the upper level formulation (UL) is the following stochastic formulation of bilevel learning:

$$\hat{\boldsymbol{\gamma}} = \underset{\boldsymbol{\gamma}\in\mathbb{F}^R}{\operatorname{argmin}} \underbrace{\mathbb{E}\left[\ell(\boldsymbol{\gamma})\right]}_{\approx \frac{1}{J}\sum_{j=1}^J \ell(\boldsymbol{\gamma}\,;\hat{\boldsymbol{x}}_j(\boldsymbol{\gamma}))} \tag{1.5}$$

$$\text{where } \hat{\boldsymbol{x}}_j(\boldsymbol{\gamma}) = \underset{\boldsymbol{x}\in\mathbb{F}^N}{\operatorname{argmin}}\, \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma},\boldsymbol{y}_j). \tag{1.6}$$

model, *i.e.*, $\boldsymbol{n}_j(\boldsymbol{A},\boldsymbol{x}_j)$. This generality is needed for applications with certain noise distributions such as Poisson noise.

The expectation, taken with respect to the training data and noise distributions, is typically approximated as a sample mean over J training examples.

The definition of bilevel methods used in (UL) is not universal in the literature. In some works, bilevel methods refer to nested optimization problems with two levels, even when the two levels result from reformulating a single-level problem, *e.g.*, [21]. That definition is much more encompassing, and includes primal-dual reformulations, Lagrangian reformulations of constrained optimization problems, and alternating methods that introduce then minimize over an auxiliary variable.

Another term in the literature, sometimes used interchangeably with a bilevel problem, is a mathematical program with equilibrium constraints (MPEC). As shown in Section 4, many bilevel optimization methods start by transforming the two-level problem into an equivalent single-level problem by replacing the lower-level optimization with a set of constraints based on optimally conditions. Bilevel problems are thus a subset of MPECs. MPECs are generally challenging due to their non-convex nature; even when the lower-level cost function is convex, the upper-level loss function is rarely convex. Importantly, $\ell(\cdot, \cdot)$ is often convex with respect to both arguments. However, $\ell(\boldsymbol{\gamma}) = \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma}))$ is generally non-convex in $\boldsymbol{\gamma}$ due to how the lower-level minimizer depends on $\boldsymbol{\gamma}$. There is a large literature on MPEC problems, *e.g.*, [8], [22], [23], and on non-convex optimization more generally [24]. Bilevel methods are one sub-field in this large literature.

1.3 Running Example

To offer a concrete example, this review will frequently refer to the following running example (Ex), a filter learning bilevel problem:

$$\hat{\boldsymbol{\gamma}} = \underset{\boldsymbol{\gamma} \in \mathbb{F}^R}{\operatorname{argmin}} \frac{1}{2} \|\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) - \boldsymbol{x}^{\text{true}}\|_2^2, \text{ where}$$

$$\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \underset{\boldsymbol{x} \in \mathbb{F}^N}{\operatorname{argmin}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + e^{\beta_0} \sum_{k=1}^{K} e^{\beta_k} \mathbf{1}' \phi_\cdot(\boldsymbol{c}_k \circledast \boldsymbol{x}; \epsilon), \tag{Ex}$$

where $\boldsymbol{\gamma} \in \mathbb{F}^R$ contains all variables that we wish to learn: the filter coefficients $\boldsymbol{c}_k \in \mathbb{F}^S$ and tuning parameters $\beta_k \in \mathbb{R}$ for all $k \in [1, K]$. We

include an auxiliary tuning parameter, $\beta_0 \in \mathbb{R}$, for easier comparison to other models. Fig. 1.2 depicts the running example and Fig. 1.3 shows example learned filters for a toy training image. Ref. [25] demonstrates how a spectral analysis of learned filters and penalty functions can be interpreted to provide insight into real-world problems.

The learnable hyperparameters can also include the sparsifying function ϕ, its corner rounding parameter ϵ, the forward model $\boldsymbol{A}$, or some aspect of the data-fit term. For example, [25], [26] learn the regularization functional and [27], [28] learn part of the forward model. Such examples are relatively rare in the bilevel methods literature to date.

Unlike many learning problems (see examples in Section 7.4), the running example (Ex) does not include any constraints on $\boldsymbol{\gamma}$. Learned filters should be those that are best at the given task, where "best" is defined by the upper-level loss function. Therefore, a zero mean or norm constraint is not generally required, though some authors have found such constraints helpful, *e.g.*, [29], [30]. Following previous

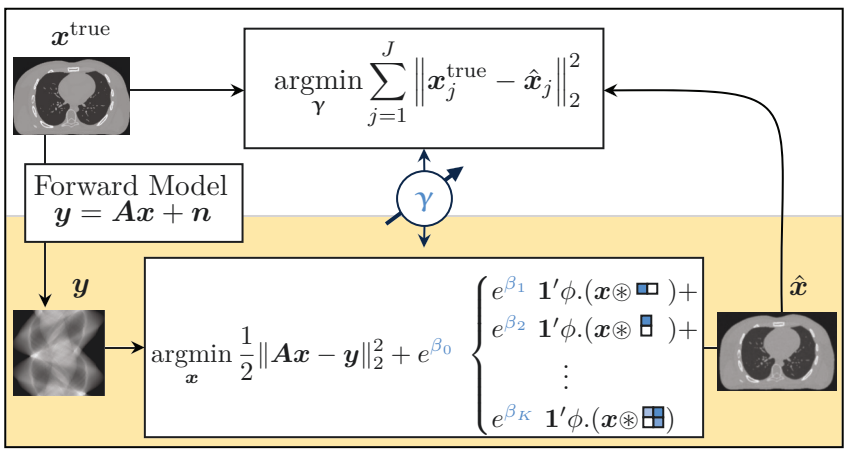


Figure 1.2: Bilevel problem in (Ex). The vector of learnable hyperparameters, $\boldsymbol{\gamma}$, includes the tuning parameters, β_k, and the filter coefficients, $\boldsymbol{c}_k$, shown as example filters. Although this review will generally consider learning filters of a single size, the figure depicts how the framework easily extends to 2d filters of different sizes.

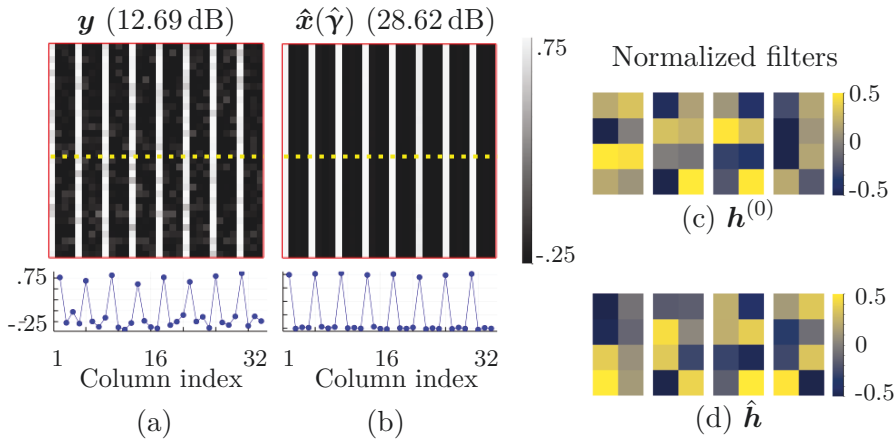


Figure 1.3: Example learned filters for a simple training image, normalized for easier visualization. The true image is zero-mean and repeats three columns of signal value -0.25 and one column of signal value 0.75. (a) Noisy image. The lower plot shows a profile of one row of the image (marked by a dotted line). The signal-to-noise ratio, as defined in (3.2), is given in parenthesis. (b) The denoised image using learned filters as in (Ex). (c) Randomly initialized filters for the bilevel method ($K = 4$ and $S = 4 \cdot 2$). (d) Corresponding learned filters. As expected based on the training image, the learned filters primarily involve vertical differences. Appendix D.1 provides further details including the regularization strength of each learned filter.

literature, *e.g.*, [31], the tuning parameters in (Ex) are written in terms of an exponential function to ensure positivity. One could re-write (Ex) without this exponentiation "trick" and then add a non-negativity constraint to the upper-level problem; most of the methods discussed in this review generalize to this common variation by substituting gradient methods for projected gradient methods.

In (Ex), we drop the sum over J training images for simplicity; the methods easily extend to multiple training signals. For ease of notation, we further simplify by considering $\boldsymbol{c}_k$ to be of length S for all k, *e.g.*, a 2D filter might be $\sqrt{S} \times \sqrt{S}$. In practice, the filters may be of different lengths with minimal impact on the methods presented in this review.

The function ϕ in (Ex) is a sparsity-promoting function. If we were to choose $\phi(z) = |z|$, then the regularizer would involve 1-norm terms

of the type common in compressed sensing formulations:

$$\mathbf{1}'\phi.(\boldsymbol{c}_k \circledast \boldsymbol{x}) = \|\boldsymbol{c}_k \circledast \boldsymbol{x}\|_1 .$$

However, to satisfy differentiability assumptions (see Section 4), this review will often consider ϕ to denote the following "corner rounded" 1-norm having the shape of a hyperbola with the corresponding first and second derivative:

$$\begin{aligned} \phi(z) &= \sqrt{z^2 + \epsilon^2} \\ \dot{\phi}(z) &= \frac{z}{\sqrt{z^2 + \epsilon^2}} \in [0, 1) \\ \ddot{\phi}(z) &= \frac{\epsilon^2}{(z^2 + \epsilon^2)^{3/2}} \in (0, \frac{1}{\epsilon}], \end{aligned} \tag{CR1N}$$

where ϵ is a small, relative to the expected range of z, parameter that controls the amount of corner rounding. (Here, we use a dot over the function rather than ∇ to indicate a derivative because ϕ has a scalar argument.)

1.4 Conclusion

Bilevel methods for selecting hyperparameters offer many benefits. Previous papers motivate them as a principled way to approach hyperparameter optimization [9], [32], as a task-based approach to learning [12], [26], [33], and/or as a way to combine the data-driven improvements from learning methods with the theoretical guarantees and explainability provided by cost function-based approaches [11], [29], [34]. A corresponding drawback of bilevel methods are their computational cost; see Sections 4 and 5 for further discussion.

The task-based nature of bilevel methods is a particularly important advantage; Section 7.4 exemplifies why by comparing the bilevel problem to single-level, non-task-based approaches for learning sparsifying filters. Task-based refers to the hyperparameters being learned based on how well they work in the lower-level cost function–the image reconstruction task in our running example. The learned hyperparameters can also adapt to the training dataset and noise characteristics. The task-based nature yields other benefits, such as making constraints or regularizers

on the hyperparameters generally unnecessary; Section 6.2 presents some exceptions and [9] further discusses bilevel methods for applications with constraints.

There are three main elements to a bilevel approach. First, the lower-level cost function in a bilevel problem defines a goal, such as image reconstruction, including what hyperparameters can be learned, such as filters for a sparsifying regularizer. Section 2 provides background on this element specifically for image reconstruction tasks, such as the one in (Ex). Section 6.1 reviews example cost functions used in bilevel methods.

Second, the upper-level loss function determines how the hyperparameters should be learned. While the squared error loss function in the running example is a common choice, Section 3 discusses other loss functions based on supervised and unsupervised image quality metrics. Section 6.2 then reviews example loss functions used in bilevel methods.

While less apparent in the written optimization problem, the third main element for a bilevel problem is the optimization approach, especially for the upper-level problem. Section 3.2 briefly discusses various hyperparameter optimization strategies, then Sections 4 and 5 present multiple gradient-based bilevel optimization strategies. Throughout the review, we refer to the running example to show how the bilevel optimization strategies apply.

2

Background: Cost Functions and Image Reconstruction

This review focuses on bilevel problems having image reconstruction as the lower-level problem. Image reconstruction involves undoing any transformations inherent in an imaging system, *e.g.*, a camera or CT scanner, and removing measurement noise, *e.g.*, thermal and shot noise, to realize an image that captures an underlying object of interest, *e.g.*, a patient's anatomy. Fig. 2.1 shows an example image reconstruction pipeline for CT data. The following sections formally define image reconstruction, discuss why regularization is important, and overview common approaches to regularization.

2.1 Image Reconstruction

Although the true object is in continuous space, image reconstruction is almost always performed on sampled, discretized signals [35]. Without going into detail of the discretization process, we define $\boldsymbol{x}^{\text{true}} \in \mathbb{F}^N$ as the "true," discrete signal. The goal of image reconstruction is to recover an estimate $\hat{\boldsymbol{x}} \approx \boldsymbol{x}^{\text{true}}$ given corrupted measurements $\boldsymbol{y} \in \mathbb{F}^N$. Although we define the signal as a one-dimensional vector for notational convenience, the mathematics generalize to arbitrary dimensions.

To find $\hat{\boldsymbol{x}}$, image reconstruction involves minimizing a cost function,

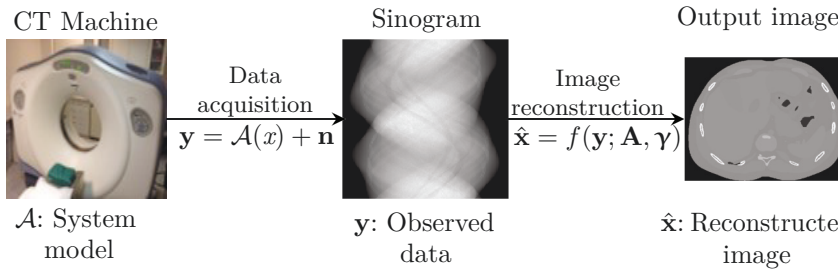


Figure 2.1: Example image reconstruction pipe-line, illustrated using XCAT phantom from [20]. Here $\mathcal{A}$ denotes the actual physical mapping of the imaging system and $\boldsymbol{A}$ denotes the numerical system matrix used for reconstruction.

$\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$, with two terms:

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}\in\mathbb{F}^N}{\operatorname{argmin}} \underbrace{\overbrace{d(\boldsymbol{x}\,;\boldsymbol{y})}^{\text{Data-fit}} + \quad \beta \overbrace{R(\boldsymbol{x}\,;\boldsymbol{\gamma})}^{\text{Regularizer}}}_{\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})} \tag{2.1}$$

The first term, $d(\boldsymbol{x}\,;\boldsymbol{y})$, is a data-fit term that captures the physics of the ideal (noiseless) system using the matrix $\boldsymbol{A} \in \mathbb{F}^{M\times N}$; that matrix models the physical system such that we expect an observation, $\boldsymbol{y}$, to be $\boldsymbol{y} \approx \boldsymbol{A}\boldsymbol{x}$.

The most common data-fit term penalizes the square Euclidean norm of the "measurement error," $d(\boldsymbol{x}\,;\boldsymbol{y}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$. This intuitive data-fit term can be derived from a maximum likelihood perspective, assuming a white Gaussian noise distribution [36]. Using the system model (1.4) and assuming the noise is normally distributed with zero-mean and variance σ^2, the maximum likelihood estimate $\hat{\boldsymbol{x}}_{\text{MLE}}$ is the image that is most likely given the observation $\boldsymbol{y}$, *i.e.*,

$$\hat{\boldsymbol{x}}_{\text{MLE}} = \underset{\boldsymbol{x}\in\mathbb{F}^N}{\operatorname{argmax}} \operatorname{Prob}(\boldsymbol{x}\,;\,\boldsymbol{y}, \sigma^2).$$

Substituting the assumed Gaussian distribution (and ignoring constants independent of $\boldsymbol{x}$),

$$\hat{\boldsymbol{x}}_{\text{MLE}} = \underset{\boldsymbol{x}\in\mathbb{F}^N}{\operatorname{argmax}}\, e^{\frac{-1}{2\sigma^2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\|^2} = \underset{\boldsymbol{x}\in\mathbb{F}^N}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 = \boldsymbol{A}^+\boldsymbol{y},$$

where $\boldsymbol{A}^+$ is the pseudo-inverse of $\boldsymbol{A}$.

The regularization term in (2.1) can be motivated by maximum *a posteriori* probability (MAP) estimation [36]. Rather than maximizing the likelihood of $\boldsymbol{x}$, the MAP estimate $\hat{\boldsymbol{x}}_{\text{MAP}}$ maximizes the conditional probability of $\boldsymbol{x}$ given the observation $\boldsymbol{y}$

$$\begin{aligned}\hat{\boldsymbol{x}}_{\text{MAP}} &= \operatorname*{argmax}_{\boldsymbol{x}\in\mathbb{F}^N} \text{Prob}(\boldsymbol{x}|\boldsymbol{y}) \\ &= \operatorname*{argmax}_{\boldsymbol{x}\in\mathbb{F}^N} \text{Prob}(\boldsymbol{y}|\boldsymbol{x})\text{Prob}(\boldsymbol{x})\end{aligned}$$

by Bayes theorem. A MAP estimator requires assuming a prior distribution on $\boldsymbol{x}$. Taking the logarithm and substituting the assumed Gaussian distribution for $\text{Prob}(\boldsymbol{y}|\boldsymbol{x}\,;\,\sigma^2)$ yields

$$\hat{\boldsymbol{x}}_{\text{MAP}} = \operatorname*{argmin}_{\boldsymbol{x}\in\mathbb{F}^N} \frac{1}{2\sigma^2}\left\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\right\|^2 - \log\left(\text{Prob}(\boldsymbol{x})\right),$$

where the regularization term in (2.1) comes from the log probability of $\boldsymbol{x}$, *i.e.*, the two are equivalent when one assumes the probability model $\text{Prob}(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\gamma})}\exp\{-R(\boldsymbol{x}\,;\,\boldsymbol{\gamma})\}$, where $Z(\boldsymbol{\gamma})$ is a scalar such that the probability integrates to one. The MLE estimate is equivalent to the MAP estimate when the prior on $\boldsymbol{x}$ is an (unbounded) "uniform" distribution.

While MAP estimation provides a useful perspective, common regularizers do not correspond to proper probability models. Further, the connection between the regularization perspective and the Bayesian perspective is simplest when the parameters $\boldsymbol{\gamma}$ are given. To learn $\boldsymbol{\gamma}$, Bayesian formulations must consider the partition function $Z(\boldsymbol{\gamma})$; that complication is avoided for bilevel formulations using a regularized lower-level problem.

Many image reconstruction problems have linear system models. In image denoising problems, one takes $\boldsymbol{A} = \boldsymbol{I}$. For image inpainting, $\boldsymbol{A}$ is a diagonal matrix of 1's and 0's, where the 0's correspond to sample indices of missing data [37]. In MRI, the system matrix is often approximated as a diagonal matrix times a discrete Fourier transform matrix, though more accurate models are often needed [38]. In some settings, one can learn $\boldsymbol{A}$ [39], or at least parts of $\boldsymbol{A}$ [40], as part of the estimation process. Although the bilevel method generalizes to learning

$\boldsymbol{A}$, the majority of papers in the field assume $\boldsymbol{A}$ is known; Section 6 discusses a few exceptions.

Using the system model (1.4), if $\boldsymbol{n}$ were known and $\boldsymbol{A}$ were invertible, we could simply compute $\hat{\boldsymbol{x}} = \boldsymbol{x}^{\text{true}} = \boldsymbol{A}^{-1}(\boldsymbol{y} - \boldsymbol{n})$. However, $\boldsymbol{n}$ is random and, while we may be able to model its characteristics, we never know it exactly. Further, the system matrix, $\boldsymbol{A}$, is often not invertible because the reconstruction problem is frequently under-determined, with fewer knowns than unknowns ($M < N$). Therefore, we must include prior assumptions about $\boldsymbol{x}^{\text{true}}$ to make the problem feasible. These assumptions about $\boldsymbol{x}^{\text{true}}$ are captured in the second, regularization term in (2.1), which depends on $\boldsymbol{\gamma}$. The following section further discusses regularizers.

In sum, image reconstruction involves finding $\hat{\boldsymbol{x}}$ that matches the collected data *and* satisfies a set of prior assumptions. The data-fit term encourages $\hat{\boldsymbol{x}}$ to be a good match for the data; without this term, there would be no need to collect data. The regularization term encourages $\hat{\boldsymbol{x}}$ to match the prior assumptions. Finally, the tuning parameter, β, controls the relative importance of the two terms. The cost function can be minimized using different optimization techniques depending on the form of each term.

This section is a very short overview of image reconstruction methods. See [7] for a more thorough review of biomedical image reconstruction.

2.2 Sparsity-Based Regularizers

The regularization, or prior assumption, term in (2.1) often involves assumptions about sparsity [3], [41]. The basic idea behind sparsity-based regularization is that the true signal is sparse in some representation, while the noise or corruption is not. Thus, one can use the representation to separate the noise and signal, and then keep only the sparse signal component. In fact, a known sparsifying representation for a signal can help to "reconstruct a signal from far fewer measurements than required by the Shannon-Nyquist sampling theorem" [41].

The regularization design problem therefore requires determining what representation best sparsifies the signal. There are two main types of sparsity-based regularizers corresponding to two representational

assumptions: synthesis and analysis [6], [36]; Fig. 2.2 depicts both. While both are popular, this review concentrates on analysis regularizers, which are more widely represented in the bilevel image reconstruction literature. This section briefly compares the analysis and synthesis formulations. Here we simplify the formulas by considering $\boldsymbol{A} = \boldsymbol{I}$; the discussion generalizes to reconstruction by including $\boldsymbol{A}$. For more thorough discussions of analysis and synthesis regularizers, see [6], [36], [42].

2.2.1 Synthesis Regularizers

Synthesis regularizers model a signal being composed of building blocks, or "atoms." Small subsets of the atoms span a low dimensional subspace and the sparsity assumption is that the signal requires using only a few of the atoms. More formally, the synthesis model is $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n}$, where the signal $\boldsymbol{x} = \boldsymbol{D}\boldsymbol{z}$ and $\boldsymbol{z}$ is a sparse vector. The columns of $\boldsymbol{D} \in \mathbb{F}^{N \times K}$ contain contain the K dictionary atoms and form a low dimensional subspace for the signal. If $\boldsymbol{D}$ is a wide matrix ($N < K$), the dictionary is over-complete and it is easier to represent a wide range of signals with a given number of dictionary atoms. The dictionary is complete when $\boldsymbol{D}$ is square (and full rank) and under-complete if $\boldsymbol{D}$ is tall (an uncommon choice).

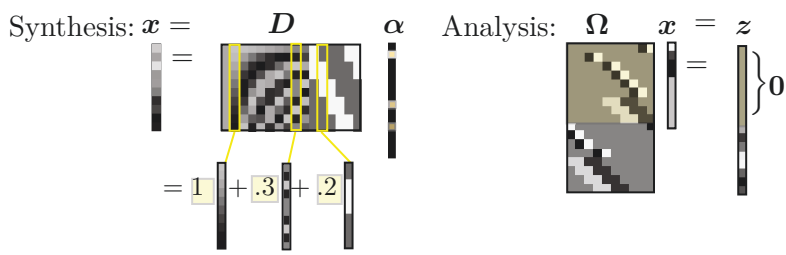


Figure 2.2: Depiction of synthesis and analysis sparsity. Under the synthesis model of sparsity (left), $\boldsymbol{x}$ is a linear combination of a few dictionary atoms. The dictionary, $\boldsymbol{D}$, is typically wide, with more atoms (columns) than elements in $\boldsymbol{x}$. Under the analysis model of sparsity (right), $\boldsymbol{x}$ is orthogonal to many filters. The filter matrix, $\boldsymbol{\Omega}$, is typically tall, with more filters (rows) than elements in $\boldsymbol{x}$.

Assuming one knows or has already learned $\boldsymbol{D}$, one can use the sparsity synthesis assumption to denoise a noisy signal $\boldsymbol{y}$ by optimizing

$$\hat{\boldsymbol{x}} = \boldsymbol{D} \cdot \underbrace{(\underset{\boldsymbol{z}\in\mathbb{F}^K}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{D}\boldsymbol{z} - \boldsymbol{y}\|^2 + \mathbf{1}'\phi.(\boldsymbol{z}))}_{\hat{\boldsymbol{z}}}. \tag{2.2}$$

The estimation procedure involves finding the sparse codes, $\hat{\boldsymbol{z}}$, from which the image is synthesized via $\hat{\boldsymbol{x}} = \boldsymbol{D}\hat{\boldsymbol{z}}$. Common sparsity-inducing functions, ϕ, are the absolute value or a non-zero indicator function, equivalent to the 1-norm and 0-norm respectively. The 2-norm is occasionally used in the regularizer, but it does not yield true sparse codes and it over-penalizes large values [43].

As written in (2.2), the synthesis formulation constrains the signal, $\boldsymbol{x}$, to be in the range of $\boldsymbol{D}$. This "strict synthesis" model can be undesirable in some applications, *e.g.*, when one is not confident in the quality of the dictionary. An alternative formulation is

$$\begin{aligned} \hat{\boldsymbol{x}} &= \underset{\boldsymbol{x}\in\mathbb{F}^N}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 + \beta R(\boldsymbol{x}), \\ R(\boldsymbol{x}) &= \min_{\boldsymbol{z}\in\mathbb{F}^K} \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|^2 + \mathbf{1}'\phi.(\boldsymbol{z}), \end{aligned} \tag{2.3}$$

which no longer constrains $\boldsymbol{x}$ to be exactly in the range of $\boldsymbol{D}$. One can also learn $\boldsymbol{D}$ while solving (2.3) [44].

Both synthesis denoising forms have equivalent sparsity constrained versions; one can replace $\mathbf{1}'\phi.(\boldsymbol{z})$ with a characteristic function that is 0 within some desired set and infinite outside it, *e.g.*,

$$\psi(\boldsymbol{z}) = \begin{cases} 0 & \text{if } \|\boldsymbol{z}\|_0 \leq \kappa \\ \infty & \text{else,} \end{cases} \tag{2.4}$$

for some sparsity constraint given by the hyperparameter $\kappa \in \mathbb{N}$.

See [43], [45] for discussions of when the synthesis model can guarantee accurate recovery of signals. The minimization problem in (2.3) is called sparse coding and is closely related to the LASSO problem [46]. One can think of the entire dictionary $\boldsymbol{D}$ as a hyperparameter that can be learned with a bilevel method [47].

2.2.2 Analysis Regularizers

Analysis regularizers model a signal as being sparsified when mapped into another vector space by a linear transformation, often represented by a set of filters. More formally, an analysis model assumes the signal satisfies $\boldsymbol{\Omega x} = \boldsymbol{z}$ for a sparse coefficient vector $\boldsymbol{z}$. Often the rows of the matrix $\boldsymbol{\Omega} \in \mathbb{F}^{K \times N}$ are thought of as filters and the rows of $\boldsymbol{\Omega}$ where $[\boldsymbol{\Omega x}]_k = 0$ span a subspace to which $\boldsymbol{x}$ is orthogonal. The analysis operator is called over-complete if $\boldsymbol{\Omega}$ is tall ($N < K$), complete if $\boldsymbol{\Omega}$ is square (and full rank), and under-complete if $\boldsymbol{\Omega}$ is wide.

A particularly common analysis regularizer is based on a discretized version of total variation (TV) [48], and uses finite difference filters (or, more generally, filters that approximate higher-order derivatives). The finite difference filters sparsify any piece-wise constant (flat) regions in the signal, leaving the edges that are often approximately sparse in natural images. Other common analysis regularizers include the discrete Fourier transform (DFT), curvelets, and wavelet transforms [49].

The literature is less consistent in analysis regularizer vocabulary, and $\boldsymbol{\Omega}$ has been called an analysis dictionary, an analysis operator, a filter matrix, and a cosparse operator. The term "cosparse" comes from the sparsity holding in the codomain of the transformation $T\{\boldsymbol{x}\} = \boldsymbol{\Omega x}$. The cosparsity of $\boldsymbol{x}$ with respect to $\boldsymbol{\Omega}$ is the number of zeros in $\boldsymbol{\Omega x}$ or $K - \|\boldsymbol{\Omega x}\|_0$ [42]. Correspondingly, "cosupport" describes the indices of the rows where $\boldsymbol{\Omega x} = 0$. We find the phrase "analysis operator" intuitive for general $\boldsymbol{\Omega}$'s and "filter matrix" more descriptive when referring to the specific (common) case when the rows of $\boldsymbol{\Omega}$ are dictated by a set of convolutional filters.

Assuming one knows, or has already learned, $\boldsymbol{\Omega}$, one can use the analysis sparsity assumption to denoise a noisy signal, $\boldsymbol{y}$, by optimizing

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in \mathbb{F}^N}{\operatorname{argmin}} \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{y}\|^2 + \beta \mathbf{1}' \phi_{\cdot}(\boldsymbol{\Omega x}). \tag{2.5}$$

An alternative version is

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in \mathbb{F}^N}{\operatorname{argmin}} \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{y}\|^2 + \beta R(\boldsymbol{x}) \tag{2.6}$$

$$R(\boldsymbol{x}) = \min_{\boldsymbol{z} \in \mathbb{F}^K} \frac{1}{2} \|\boldsymbol{\Omega x} - \boldsymbol{z}\|^2 + \mathbf{1}' \phi_{\cdot}(\boldsymbol{z}).$$

As in the synthesis case, both analysis formulations have equivalent sparsity-constrained forms using a characteristic function as in (2.4).

See [49] for an error bound on the estimated signal $\hat{\boldsymbol{x}}$ when using a 1-norm as the regularization function.

2.2.3 Comparing Analysis and Synthesis Approaches

The analysis and synthesis models are equivalent when the dictionary and analysis operator are invertible, with $\boldsymbol{D} = \boldsymbol{\Omega}^{-1}$ [36]. Furthermore, in the denoising scenario where the system matrix $\boldsymbol{A}$ is identity, the two are almost equivalent in the under-complete case, with the lack of full equivalence stemming from the analysis form not constraining $\boldsymbol{x}$ to be in the range space $\boldsymbol{D}$ [36].

As shown in [41, Example 3.1], the analysis model can more generally be related to a Lasso-like problem using Legendre-Fenchel conjugates and convex duality. Appendix A briefly reviews duality and the main results from primal-dual analysis used throughout this review. Considering the analysis operator learning problem (2.5), when the sparsity promoting function ϕ is convex and $\phi(z) < \infty$ for some z, the dual problem corresponding to (2.5) is

$$\hat{\boldsymbol{d}} = \underset{\boldsymbol{d} \in \mathbb{F}^K}{\operatorname{argmin}} \frac{1}{2} \left\| \boldsymbol{\Omega}' \boldsymbol{d} - \boldsymbol{y} \right\|^2 + \phi^*(\boldsymbol{d}),$$

where $\boldsymbol{d}$ is the dual variable and ϕ^* is the conjugate function of ϕ. (The primal solution $\hat{\boldsymbol{x}}$ can be computed from $\hat{\boldsymbol{d}}$ using (A.11).) This dual problem is similar in form to the inner minimization in the strict synthesis formulation (2.2). This relation between the analysis model and its dual formulation is limited to cases where ϕ is convex.

Whether analysis-based or synthesis-based regularizers are generally preferable is an open question, and the answer likely depends on the application and the relative importance of reconstruction accuracy and speed [36]. Synthesis regularization is perhaps easier to interpret because of its generative nature. In contrast, bilevel analysis filter learning is a discriminative learning approach: the task-based filters must learn to distinguish "good" and "bad" image features.

The synthesis approach used to be "widely considered to provide superior results" [36, p. 950]. However, [36] goes on to show that an

analysis regularizer produced more accurate reconstructed images in experiments on real images. Later analysis-based results also show competitive, if not superior, quality results when compared to similar synthesis models [50], [51]. See [52] for a survey of optimization methods for MRI reconstruction and a comparison of the computational challenges for cost functions with synthesis and analysis-based regularizers.

The analysis and synthesis regularizers in (2.2) and (2.6) quickly yield infeasibly large operators as the signal size increases. In practice, both approaches are usually implemented with patch-based formulations. For the synthesis approach, the patches typically overlap and there is an averaging effect. Analysis regularizers that have rows corresponding to filters, called the convolutional analysis model, extend very naturally to a global image regularizer. For example, in the lower-level cost function of our running filter learning example (Ex), we can define an analysis regularizer matrix as follows:

$$\mathbf{\Omega} = \begin{bmatrix} \boldsymbol{C}_1 \\ \vdots \\ \boldsymbol{C}_K \end{bmatrix} \in \mathbb{F}^{KN \times N}. \tag{2.7}$$

Imposing this convolutional structure on $\mathbf{\Omega}$ helps make learning problems feasible as one only has to learn the S coefficients of each of the K filters rather than learning the full $\mathbf{\Omega}$ matrix. This structure also ensures translation invariance of the regularizer. See [30] and [53] for discussion of the connections between global models and patch-based models for analysis regularizers. The running example in this survey focuses on bilevel learning of convolutional analysis regularizers.

2.3 Brief History of Analysis Regularizer Learning

In 2003, Haber and Tenorio [26] proposed using bilevel methods to learn part of the regularizer in inverse problems. The authors motivate the use of bilevel methods through the task-based nature, noting that "the choice of good regularization operators strongly depends on the forward problem." They consider learning tuning parameters, space-varying weights, and regularization operators (comparable to defining ϕ), all

for regularizers based on penalizing the energy in the derivatives of the reconstructed image. Their framework is general enough to handle learning filters. Ref. [26] was published a few years earlier than the other bilevel methods we consider in this review and was not cited in most other early works; [54] calls it a "groundbreaking, but often overlooked publication."

In 2005, Roth and Black [55] proposed the Field of Experts (FoE) model to learn filters. Although the FoE is not formulated as a bilevel method, many papers on bilevel methods for filter learning cite FoE as a starting or comparison point. The FoE model is a translation-invariant analysis operator model, built on convolutional filters. It is motivated by the local operators and presented as a Markov random field model, with the order of the field determined by the filter size.

Under the FoE model, the negative log[1] of the probability of a full image, $\boldsymbol{x}$, is proportional to

$$\sum_k \beta_k \, \phi.(\boldsymbol{c}_k \circledast \boldsymbol{x}) \text{ where } \phi(z) = \log\left(1 + \frac{1}{2}z^2\right). \tag{2.8}$$

This (non-convex) choice of sparsity function ϕ stems from the Student-t distribution. Ref. [55] learns the filters and filter-dependent tuning parameters such that the model distribution is as close as possible (defined using Kullback-Leibler divergence) to the training data distribution.

In 2007, Tappen, Liu, Adelson, *et al.* [56] proposed a different model based on convolutional filters: the Gaussian Conditional Random Field (GCRF) model. Rather than using a sparsity promoting regularizer, the GCRF uses a quadratic function for ϕ. The authors introduce space-varying weights, $\boldsymbol{W}$, so that the quadratic model does not overly penalize sharp features in the image. The general idea behind $\boldsymbol{W}$ is to use the given (noisy) image to guess where edges occur, and correspondingly penalize those areas less to avoid blurring edges. The likelihood for GCRF model is thus (to within a proportionality constant

[1] By taking the log of the probability model in [55], the connection between the FoE and the regularization term in the lower-level of the running filter learning example (Ex) is more evident.

and monotonic function transformations):

$$\sum_k \|\boldsymbol{c}_k \circledast \boldsymbol{x} - e_k\{\boldsymbol{x}\}\|^2_{\boldsymbol{W}_k},$$

where the term $e_k\{\boldsymbol{x}\}$ captures the estimated value of the filtered image. For example, [56] used one averaging filter and multiple differencing filters for the $\boldsymbol{c}_k$'s. The corresponding estimated values are $\boldsymbol{x}$ for the averaging filter and zero for the differencing filters.

The filters, $\boldsymbol{c}_k$, are pre-determined in the GCRF model; the learned element is how to form the weights as a function of image features. Specifically, each $\boldsymbol{W}_k$ is formed as a linear combination of the (absolute) responses to a set of edge-detecting filters, with the linear combination coefficients learned from training data. Rather than maximizing the likelihood of training data as in [55], [56] learns these coefficients to minimize the (corner-rounded) l_1 norm of the error of the predicted image, which is a form of bilevel learning even though not described with that terminology.

Apparently one of the first papers to explicitly propose using bilevel methods to learn filters appeared in 2009, where Samuel and Tappen [31] considered a bilevel formulation where the upper-level loss was the squared Euclidean norm of training data and the lower-level cost was a denoising task based on filter sparsity equivalent to (Ex). The method builds on the FoE model, using the same ϕ as in [55], but now learning the filters using a bilevel formulation rather than by maximizing a likelihood.

In 2011, Peyré and Fadili [33] proposed a similar bilevel method to learn analysis regularizers. The authors generalized the denoising task to use an analysis operator matrix and a wider class of sparsifying functions. Their results concentrate on the convolutional filter case with a corner-rounded 1-norm for ϕ.

Both [31] and [33] focus on introducing the bilevel method for analysis regularizer learning, with denoising or inpainting as illustrations. Section 4 further discusses the methodology of both papers. Many of the bilevel based papers in this review build on one or both of their efforts. The rest of the review will summarize other bilevel based papers; here, we highlight some of papers in the non-bilevel thread of the literature

for context and comparison.

Ophir, Elad, Bertin, *et al.* [57] proposed another approach to learning an analysis operator. The method learns the operator one row at a time by searching for vectors orthogonal to the training signals. Algorithm parameters were chosen empirically without an upper-level loss function as a guide.

Between 2011 [58] and 2013 [59], Yaghoobi, Nam, Gribonval, and Davies were among the first to formally present analysis operator learning as an optimization problem. Their conference paper [58] considered noiseless training data and proposed learning an analysis operator as

$$\underset{\boldsymbol{\Omega}}{\operatorname{argmin}} \|\boldsymbol{\Omega}\boldsymbol{X}^{\text{true}}\|_1 \text{ s.t. } \boldsymbol{\Omega} \in \mathcal{S} \tag{2.9}$$

for some constrained set $\mathcal{S}$. Each column of $\boldsymbol{X}^{\text{true}} \in \mathbb{F}^{N\times J}$ contains a training sample. The authors discussed varying options for $\mathcal{S}$, including a row norm, full rank, and tight frame constrained set.

Without any constraint on $\boldsymbol{\Omega}$, the trivial solution to (2.9) would be to learn the zero matrix, which is not informative for any problem such as image denoising. Section 7.4 discusses in more detail the need for constraints and the various constraint options proposed for filter learning.

Ref. [59] extends (2.9) to the noisy case where one does not have access to $\boldsymbol{X}^{\text{true}}$. The proposed cost function is

$$\underset{\boldsymbol{\Omega},\boldsymbol{X}}{\operatorname{argmin}} \|\boldsymbol{\Omega}\boldsymbol{X}\|_1 + \frac{\beta}{2}\|\boldsymbol{X}-\boldsymbol{Y}\|^2 \text{ s.t. } \boldsymbol{\Omega} \in \mathcal{S}, \tag{2.10}$$

where each column of $\boldsymbol{Y}$ contains a noisy data vector. Ref. [59] minimized (2.10) by alternating updating $\boldsymbol{X}$, using alternating direction method of multipliers (ADMM), and $\boldsymbol{\Omega}$, using a projected subgradient method for various constraint sets $\mathcal{S}$, especially Parseval tight frames.

In the same time-frame, Kunisch and Pock [60] started to analyze the theory behind the bilevel problem, building off the ideas in [31], [33]. Among the theoretical analysis, [60] proves the existence of upper-level minimizers when the bilevel problem takes the form of (Ex), $\boldsymbol{\gamma}$ is the tuning parameters (the β_k values), and ϕ corresponds to the squared 2-norm or the 1-norm. When $\phi(z) = z^2$, there is an analytic

solution to the lower-level problem and a corresponding closed-form solution to the gradient of the upper-level problem; [60] uses this fact to discuss qualitative properties of the minimizer. Ref. [60] also proposed an efficient semi-smooth Newton algorithm for finding $\hat{\boldsymbol{\gamma}}$ (using corner rounding for the 1-norm case) and used this algorithm to make empirical comparisons of multiple sparsifying functions (2-norm, 1-norm, and $p = 1/2$-norm) and different pre-defined filter banks.

Also in 2013, Ravishankar and Bresler [51] made a distinction between the analysis model, where one models $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n}$ with $\boldsymbol{z} = \boldsymbol{\Omega x}$ being sparse, and the transform model, where $\boldsymbol{\Omega y} = \boldsymbol{z} + \boldsymbol{n}$ where $\boldsymbol{z}$ is sparse. The analysis version models the measurement as being a cosparse signal plus noise; the transform version models the measurement as being approximately cosparse. Another perspective on the distinction is that, if there is no noise, the analysis model constrains $\boldsymbol{y}$ to be in the range space of $\boldsymbol{\Omega}$, while there is no such constraint on the transform model. The corresponding transform learning problem is

$$\underset{\boldsymbol{\Omega}}{\operatorname{argmin}} \min_{\boldsymbol{Z}} \|\boldsymbol{\Omega Y} - \boldsymbol{Z}\|_2^2 + R(\boldsymbol{\Omega}) \quad \text{s.t. } \|\boldsymbol{Z}_i\|_0 \leq \alpha \; \forall i, \tag{2.11}$$

where i indexes the columns of $\boldsymbol{Z}$. Ref. [51] considers only square matrices $\boldsymbol{\Omega}$. The regularizer, R, promotes diversity in the rows of $\boldsymbol{\Omega}$ to avoid trivial solutions, similar to the set constraint in (2.10).

A more recent development is directly modeling the convolutional structure during the learning process. In 2020, [61] proposed Convolutional Analysis Operator Learning (CAOL) to learn convolutional filters without patches. The CAOL cost function is

$$\underset{[\boldsymbol{c}_1,\ldots,\boldsymbol{c}_K]}{\operatorname{argmin}} \sum_{k=1}^{K} \min_{\boldsymbol{z}} \frac{1}{2} \|\boldsymbol{c}_k \circledast \boldsymbol{x} - \boldsymbol{z}\|_2^2 + \beta \|\boldsymbol{z}\|_0 \text{ s.t. } [\boldsymbol{c}_1 \ldots \boldsymbol{c}_K] \in \mathcal{S}. \tag{2.12}$$

Unlike the previous cost functions, which typically require patches, CAOL can easily handle full-sized training images $\boldsymbol{x}$ due to the nature of the convolutional operator.

While model-based methods were being developed in the signal processing literature, convolutional neural network (CNN) models were being advanced and trained in the machine learning and computer vision literature [62] [63] [64]. The filters used in CNN models like

U-Nets [65] can be thought of as having analysis roles in the earlier layers, and synthesis roles in the final layers [66]. See also [67] for further connections between analysis and transform models within CNN models. CNN training is usually supervised, and the supervised approach of bilevel learning of filters strengthens the relationships between the two approaches. A key distinction is that CNN models are generally feed-forward computations, whereas bilevel methods of the form (LL) have a cost function formulation. See Section 7 for further discussion of the parallels between CNNs and bilevel methods.

2.4 Summary

This background section focused on the lower-level problem: image reconstruction with a sparsity-based regularizer. After defining the problem and the need for regularization, Section 2.3 reviewed the history of analysis regularizer learning and included many examples of methods to learn hyperparameters.

Bilevel methods are just one, task-based way to learn such hyperparameters. Section 7.4 further expands on this point, but we can already see benefits of the task-based nature of bilevel methods. Without the bilevel approach, filters are often learned such that they best sparsify training data. These sparsifying filters can then be used in a regularizer for image reconstruction tasks. However, they are learned to *sparsify*, not necessarily to best *reconstruct*. In contrast, the bilevel approach aims to learn filters that best reconstruct images (or whatever other task is desired), even if those filters are not the ones that best sparsify. Although this distinction may seem subtle, [68] shows that different filters work better for image denoising versus image inpainting.

Having provided some background on the lower-level cost function and motivated bilevel methods, this review now turns to defining the upper-level loss function and surveying methods of hyperparameter optimization.

3

Background: Loss Functions and Hyperparameter Optimization

Most inverse problems involve at least one hyperparameter. For example, the general reconstruction cost function (2.1) requires choosing the tuning parameter β that trades-off the influence of the data-fit and regularization terms. The field of hyperparameter optimization is large and encompasses categorical hyperparameters, such as which optimizer to use; conditional hyperparameters, where certain hyperparameters are relevant only if others take on certain values; and integer or real-valued hyperparameters [69]. Here, we focus on learning real-valued, continuous hyperparameters.

A hyperparameter's value can greatly influence the properties of the minimizer and a tuned hyperparameter typically improves over a default setting [69]. Fig. 3.1 illustrates how changing a tuning parameter can dramatically impact the visual quality of the reconstructed image. If β is too low, not enough weight is on the regularization term, and the minimizer is likely to be corrupted by noise in the measurements. If β is too high, the regularization term dominates, and the minimizer will not align with the measurements.

Generalizing to an arbitrary learning problem that could have multiple hyperparameters, the goal of hyperparameter optimization is to

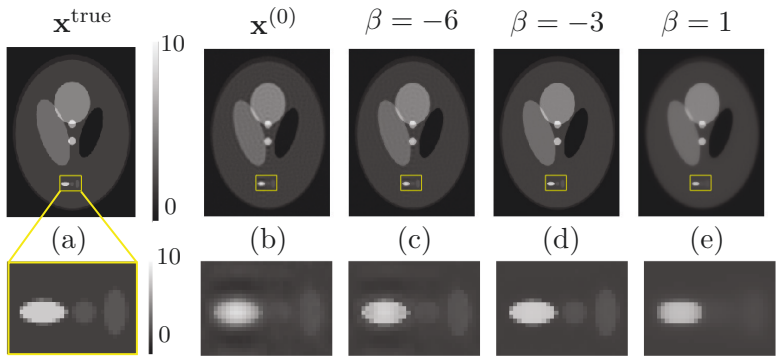


Figure 3.1: Example reconstructed simulated MRI images that demonstrate the importance of tuning parameters. (a) The original image, $\boldsymbol{x}^{\text{true}} \in R^N$, is a SheppLogan phantom [70] and N is the number of pixels. (b) A simplistic reconstruction $\frac{1}{N}\boldsymbol{A}'\boldsymbol{y}$ of the noisy, undersampled data, $\boldsymbol{y}$. This image is used as initialization, $\boldsymbol{x}^{(0)}$, for the following reconstructions. (c-e) Reconstructed images, found by optimizing $\operatorname{argmin}_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\|_2^2+10^{\beta}N\phi(\boldsymbol{C}\boldsymbol{x})$, where $\boldsymbol{C}$ is an operator that takes vertical and horizontal finite differences. The reconstructed images correspond to (c) $\beta = -6$, resulting in an image that contains ringing artifacts, (d) $\beta = -3$, resulting in a visually appealing $\hat{\boldsymbol{x}}$, and (e) $\beta = 1$, resulting in a blurred image. The demonstration code and more details about the reconstruction set-up are available on github [71].

find the "best" set of hyperparameters, $\hat{\boldsymbol{\gamma}}$, to meet a goal, described by a loss function ℓ. Specifically, we wish to solve

$$\hat{\boldsymbol{\gamma}} = \underset{\boldsymbol{\gamma}\in\Gamma}{\operatorname{argmin}}\, \mathbb{E}\left[\ell(\boldsymbol{\gamma})\right], \tag{3.1}$$

where Γ is the set of all possible hyperparameters and the expectation is taken with respect to the distribution of the input data. If evaluating ℓ uses the output of another optimization problem, *e.g.*, $\hat{\boldsymbol{x}}$, then (3.1) is a bilevel problem as defined in (UL).

There are two key tasks in hyperparameter optimization.

1. The first is to quantify how good a hyperparameter is; this step is equivalent to defining ℓ in (3.1). Section 3.1 focuses on a high-level discussion of loss functions in the broader image quality assessment (IQA) literature. Section 6.2 builds on this discussion

by reviewing specific loss functions used in bilevel methods.

2. The second step is finding a good hyperparameter, which is equivalent to designing an optimization algorithm to minimize (3.1). Section 3.2 introduces common approaches, all of which have computational requirements that scale at least linearly with the number of hyperparameters. This scaling quickly becomes infeasible for large $\boldsymbol{\gamma}$, which motivates the focus on gradient-based bilevel methods in the remainder of this review.

The next two sections address each of these tasks in turn.

3.1 Image Quality Metrics

This section concentrates on the part of the upper-level loss function that compares the reconstructed image, $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$, to the true image, $\boldsymbol{x}^{\text{true}}$. As mentioned in Section 1, bilevel methods rarely require additional regularization for $\boldsymbol{\gamma}$, but it is simple to add a regularization term to any of the loss functions if useful for a specific application. To discuss only the portion of the loss function that measures image quality, we use the notation $\ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) = l(\hat{\boldsymbol{x}},\, \boldsymbol{x}^{\text{true}})$.

Picking a loss function is part of the engineering design process. No single loss function is likely to work in all scenarios; users must decide on the loss function that best fits their system, data, and goals. Consequently, there are a wide variety of loss functions proposed in the literature and some approaches combine multiple loss functions [5], [72].

One important decision criteria when selecting a loss function is the end purpose of the image. Much of the IQA literature focuses on metrics for images of natural scenes and is often motivated by applications where human enjoyment is the end-goal [73], [74]. In contrast, in the medical image reconstruction field, image quality is not the end-goal, but rather a means to achieving a correct diagnosis. Thus, the perceptual quality is less important than the information content.

There are two major classes of image quality metrics in the IQA literature, called full reference and no reference IQA[1]. The principles

[1]There are also reduced-reference image quality metrics, but we will not consider those here.

are somewhat analogous to supervised and unsupervised approaches in the machine learning literature. This section discusses some of the most common full reference and no reference loss functions; see [75] for a comparison of 11 full-reference IQA metrics and [76] for additional no-reference IQA metrics.

Perhaps surprisingly, the bilevel filter learning literature contains few examples of loss functions other than squared error or slight variants (see Section 6.2). While this is likely at least partially due to the computational requirements of bilevel methods (see Section 4 and 5), exploring additional loss functions is an interesting future direction for bilevel research.

3.1.1 Full Reference IQA

Full reference IQA metrics assume that you have a noiseless image, $\boldsymbol{x}^{\text{true}}$, for comparison. Some of the simplest (and most common) full reference loss functions are:

- Mean squared error (MSE or ℓ_2 error):

$$l_{\text{MSE}}(\hat{\boldsymbol{x}}, \boldsymbol{x}^{\text{true}}) = \frac{1}{N} \left\| \hat{\boldsymbol{x}} - \boldsymbol{x}^{\text{true}} \right\|_2^2$$

- Mean absolute error (or ℓ_1 error): $l_{\text{MAE}}(\hat{\boldsymbol{x}}, \boldsymbol{x}^{\text{true}}) = \frac{1}{N} \left\| \hat{\boldsymbol{x}} - \boldsymbol{x}^{\text{true}} \right\|_1$
- Signal to Noise Ratio (SNR, commonly expressed in dB):

$$l_{\text{SNR}}(\hat{\boldsymbol{x}}, \boldsymbol{x}^{\text{true}}) = 10\log\left(\frac{\left\| \boldsymbol{x}^{\text{true}} \right\|_2^2}{\left\| \hat{\boldsymbol{x}} - \boldsymbol{x}^{\text{true}} \right\|_2^2} \right) \tag{3.2}$$

- Peak SNR (PSNR, in dB): $l_{\text{PSNR}}(\hat{\boldsymbol{x}}, \boldsymbol{x}^{\text{true}}) = 10\log\left(\frac{N\|\boldsymbol{x}^{\text{true}}\|_\infty}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}^{\text{true}}\|_2^2} \right)$.

The Euclidean norm is also frequently used as the data-fit term for reconstruction.

MSE (and the related metrics SNR and PSNR) are common in the signal processing field; they are intuitive and easy to use because they are differentiable and operate point-wise. However, these measures do not align well with human perceptions of image quality [75], [77]. For example, scaling an image by 2 leads to the same visual quality but causes 100% MSE. Fig. 3.2 shows a clean image and five images with

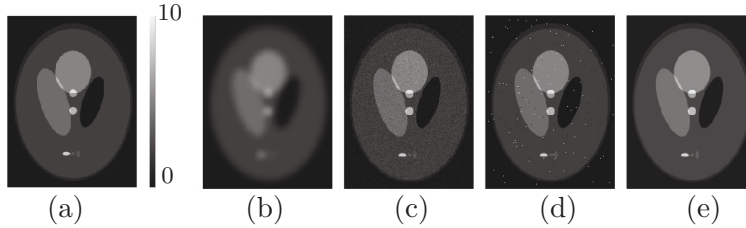


Figure 3.2: Example distortions that yield images with identical normalized squared error values: $\|\boldsymbol{x}^{\text{true}} - \boldsymbol{x}\| / \|\boldsymbol{x}^{\text{true}}\| = 0.17$. (a) The original image, $\boldsymbol{x}^{\text{true}}$, is a SheppLogan phantom [70]. The remaining images are displayed with the same colormap and have the following distortions: (b) blurred with an averaging filter, (c) additive, white Gaussian noise, (d) salt and pepper noise, and (e) a constant value added to every pixel.

different degradations. All five degraded images have almost equivalent squared errors, but humans judge their qualities as very different.

Tuning parameters using MSE as the loss function tends to lead to images that are overly-smoothed, sacrificing high frequency information [78], [79]. High frequency details are particularly important for perceptual quality as they correspond to edges in images. Therefore, some authors use the MSE on edge-enhanced versions of images to discourage solutions that blur edges. For example, [80] used a "high frequency error norm" metric consisting of the MSE of the difference of $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}^{\text{true}}$ after applying a Laplacian of Gaussian (LoG) filter.

Another common full-reference IQA is Structural SIMilarity (SSIM) [81] that attempts to address the issues with MSE discussed above. SSIM is defined in terms of the local luminance, contrast, and structure in images. A multiscale extension of SSIM, called MS-SSIM, considers these features at multiple resolutions [82]. The method computes the contrast and structure measures of SSIM for downsampled versions of the input images and then defines MS-SSIM as the product of the luminance at the original scale and the contrast and structure measures at each scale. However, SSIM and MS-SSIM may not correlate well with human observer performance on radiological tasks [83].

Recent works, *e.g.*, [76], [84], consider using (deep) CNN models for IQA. CNN methods are increasingly popular and their use as a model

for the human visual system [85] makes them an attractive tool for assessing images. For example, [84] proposed a CNN with convolutional and pooling layers for feature extraction and fully connected layers for regression. They used VGG [86], a frequently-cited CNN design with 3×3 convolutional kernels, as the basis of the feature extraction portion of their network. Ref. [84] showed that deeper networks with more learnable parameters were able to better predict image quality. However, datasets of images with quality labels remain relatively scarce, making it difficult to train deep networks.

3.1.2 No Reference IQA

No reference, or unsupervised, IQA metrics attempt to quantify an image's quality without access to a noiseless version of the image. These metrics rely on modeling statistical characteristics of images or noise. Many no reference IQA metrics assume the noise distribution is known.

The discrepancy principle is a classic example of an IQA metric that uses an assumed noise distribution to characterize the expected relation between the reconstructed image and the noisy data. For additive zero-mean white Gaussian noise with known variance σ^2, the discrepancy principle uses the fact that the expected MSE in the data space is the noise variance [18]:

$$\mathbb{E}\left[\frac{1}{M}\left\|\boldsymbol{A}\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) - \boldsymbol{y}\right\|_2^2\right] = \sigma^2.$$

The discrepancy principle can be used as a stopping criteria in machine learning methods or as a loss function, *e.g.*,

$$\ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) = \left(\frac{1}{M}\left\|\boldsymbol{A}\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) - \boldsymbol{y}\right\|_2^2 - \sigma^2\right)^2.$$

However, images of varying quality can yield the same noise estimate, as seen in Fig. 3.2. Related methods have been developed for Poisson noise as well [87].

Paralleling MSE's popularity among supervised loss metrics, Stein's Unbiased Risk Estimator (SURE) [88] is an unbiased estimate of MSE that does not require noiseless images. Let $\boldsymbol{y} = \boldsymbol{x}^{\text{true}} + \boldsymbol{n}$ denote a signal plus noise measurement where $\boldsymbol{n}$ is, as above, Gaussian noise

with known variance σ^2. The SURE estimate of the MSE of a denoised signal, $\hat{\boldsymbol{x}}$, is

$$\frac{1}{N}\left\|\hat{\boldsymbol{x}}(\boldsymbol{y})-\boldsymbol{y}\right\|_2^2-\sigma^2+\frac{2\sigma^2}{N}\mathrm{Tr}\left(\nabla_y\hat{\boldsymbol{x}}(\boldsymbol{y})\right), \tag{3.3}$$

where we write $\hat{\boldsymbol{x}}$ as a function of $\boldsymbol{y}$ to emphasize the dependence and $\mathrm{Tr}(\cdot)$ denotes the trace operation. For large signal dimensions N, such as is common in image reconstruction problems, the law of large numbers suggests SURE is a fairly accurate approximation of the true MSE.

It is often impractical to evaluate the divergence term in (3.3), due to computational limitations or not knowing the form of $\hat{\boldsymbol{x}}(\boldsymbol{y})$. A Monte-Carlo approach to estimating the divergence [89] uses the following key equation:

$$\mathrm{Tr}\left(\nabla_{\boldsymbol{y}}\hat{\boldsymbol{x}}(\boldsymbol{y})\right)=\lim_{\epsilon\to 0}\mathbb{E}\left[\boldsymbol{b}'\cdot\frac{\hat{\boldsymbol{x}}(\boldsymbol{y}+\epsilon\boldsymbol{b})-\hat{\boldsymbol{x}}(\boldsymbol{y})}{\epsilon}\right], \tag{3.4}$$

where $\boldsymbol{b}$ is a independent and identically distributed (i.i.d.) random vector with zero mean, unit variance, and bounded higher order moments. Theoretical and empirical arguments show that a single noise vector can well-approximate the divergence [89], so only two calls to the lower-level solver $\hat{\boldsymbol{x}}(\boldsymbol{y})$ are required. This method treats the lower-level problem like a blackbox, thus allowing one to estimate the divergence of complicated functions, including those that may not be differentiable.

See [90]–[92] for examples of applying the Monte-Carlo estimation of SURE to train deep neural networks, and [93], [94] for two examples of learning a tuning parameter using a bilevel approach with SURE as the upper-level loss function. For extensions to inverse problems (where $\boldsymbol{A}\neq\boldsymbol{I}$) and to noise from exponential families, see [95]–[97].

While SURE and the discrepancy principle are popular no-reference metrics in the signal processing literature, there are many additional no-reference metrics in the image quality assessment literature. These metrics typically depend on modeling one (or more) of three things [74]:

- image source characteristics,
- image distortion characteristics, *e.g.*, blocking artifact from JPEG compression, and/or
- human visual system perceptual characteristics.

As an example of a strategy that can capture both image source and human visual system characteristics, natural scene[2] statistics characterize the distribution of various features in natural scenes, typically using some filters [74], [98]. If a feature reliably follows a specific statistical pattern in natural images but has a noticeably different distribution in distorted images, one can use that feature to assign quality scores to images. Some IQA metrics attempt to first identify the type of distortion and measure features specific to that distortion, while others use the same features for all images.

In addition to their use in full-reference IQA, CNN models have be trained to perform no-reference IQA [84], [99]. For example, [99] proposes a CNN model that extracts small (32×32) patches from images, estimates the quality of each one, and averages the scores over all patches to get a quality score for the entire image. Briefly, their method involves local contrast normalization for each patch, applying (learned) convolutional filters to extract features, maximum and minimum pooling, and fully connected layers with rectified linear units (ReLUs). As with most no reference IQAs, [99] trained their CNN on a dataset of human encoded image quality scores (see [100] for a commonly used collection of publicly available test images with quality scores). Unlike most other IQA approaches, [99] used backpropagation to learn all the CNN weights rather than learning a transformation from handcrafted features to quality scores.

Interestingly, some of the no-reference IQA metrics [74], [98], [99] approach the performance of the full-reference IQAs in terms of their ability to match human judgements of image quality. This observation suggests that there is room to improve full-reference IQA metrics and that assessing image quality is a very challenging problem!

3.2 Parameter Search Strategies

After selecting a metric to measure how good a hyperparameter is, the next task is devising a strategy to find the best hyperparameter according to that metric. Search strategies fall into three main categories: (i)

[2]Natural scenes are those captured by optical cameras (not created by computer graphics or other artificial processes) and are not limited to outdoor scenes.

model-free, ℓ-only; (ii) model-based, ℓ-only; and (iii) gradient-based, using both ℓ and $\nabla\ell$. Model-free strategies do not assume any information about about the hyperparameter landscape, whereas model-based strategies use historical ℓ evaluations to predict the loss-function at untested hyperparameter values.

The following sections describe common model-free and model-based hyperparameter search strategies that only use ℓ. See [9, Ch. 13 and Ch. 20.6] for discussion of additional gradient-free methods for bilevel problems, *e.g.*, population-based evolutionary algorithms, and [101] for a general discussion of derivative-free optimization methods.

The third class of hyperparameter optimization schemes are approaches based on gradient descent of a bilevel problem. The high-level strategy in bilevel approaches is to calculate the gradient of the upper-level loss function ℓ with respect to $\boldsymbol{\gamma}$ and then use any gradient descent method to minimize $\boldsymbol{\gamma}$. Although this approach can be computationally challenging, it generalizes well to a large number of hyperparameters. Section 4 and Section 5 discuss this point further and go into depth on different methods for computing this gradient.

3.2.1 Model-free Hyperparameter Optimization

The most common search strategy is probably an empirical search, where a researcher tries different hyperparameter combinations manually. A punny, but often accurate, term for this manual search is GSD: grad[uate] student descent [102]. Bergstra and Bengio [103] hypothesizes that manual search is common because it provides some insight as the user must evaluate each option, it requires no overhead for implementation, and it can perform reliably in very low dimensional hyperparameter spaces.

Grid search is a more systematic alternative to manual search. When there are only one or two continuous hyperparameters, or the possible set of hyperparameters, $\boldsymbol{\Gamma}$, is small, a grid search (or exhaustive search) strategy may suffice to find the optimal value, $\hat{\boldsymbol{\gamma}}$, to within the grid spacing. However, the complexity of grid search grows exponentially with the number of hyperparameters. Regularizers frequently have many hyperparameters, so one generally requires a more sophisticated search

strategy.

One popular approach is random search, which [103] shows is superior to a grid search, especially when some hyperparameters are more important than others. There are also variations on random search, such as using Poisson disk sampling theory to explore the hyperparameter space [104]. The simplicity of random search makes it popular, and, even if one uses a more complicated search strategy, random search can provide a useful baseline or an initialization strategy. However, random search, like grid search, suffers from the curse of dimensionality, and is less effective as the hyperparameter space grows.

Another group of model-free blackbox strategies are population-based methods such as evolutionary algorithms. A popular population-based method is the covariance matrix adaption evolutionary strategy (CMA-ES) [105]. In short, every iteration, CMA-ES involves sampling a multivariate normal distribution to create a number of "offspring" samples. Mimicking natural selection, these offspring are judged according to some fitness function, a parallel to the upper-level loss function. The fittest offspring determine the update to the normal distribution and thus "pass on" their good characteristics to the next generation.

3.2.2 Model-based Hyperparameter Optimization

Model-based search strategies assume a model (or prior) for the hyperparameter space and use only loss function evaluations (no gradients). This section discusses two common model-based strategies: Bayesian methods and trust region methods.

Bayesian methods fit previous hyperparameter trials' results to a model to select the hyperparameters that appear most promising to evaluate next [106]. For example, a common model for the hyperparameters is the Gaussian Process prior. Given a few hyperparameter and cost function points, a Bayesian method involves the following steps.

1. Find the mean and covariance functions for the Gaussian Process. The mean function will generally interpolate the sampled points. The covariance function is generally expressed as a kernel function, often using squared exponential functions [107].
2. Create an acquisition function. The acquisition function captures

how desirable it is to sample ("acquire") a hyperparameter setting. Thus, it should be large (desirable) for hyperparameter values that are predicted to yield small loss function values or that have high enough uncertainty that they may yield low losses. The design of the acquisition function thus trades-off between exploring new areas of the hyperparameter landscape with high uncertainty and a more locally focused exploitation of the current best hyperparameter settings. See [107] for a discussion of specific acquisition function designs.

3. Maximize the acquisition function (typically designed to be easy to optimize) to determine which hyperparameter point to sample next.
4. Evaluate the loss function at the new hyperparameter candidate.

These steps repeat for a given amount of time or until convergence.

The derivative-free, trust-region method (TRM) [108] is similar to Bayesian optimization in that it involves fitting an easier to optimize function to the loss function of interest, ℓ, and then minimizing the easier, surrogate function (the "model"). The "trust-region" in TRM captures how well the model matches the observed ℓ values and determines the maximum step at every iteration, typically by comparing the actual decrease in ℓ (based on observed function evaluations) to the predicted decrease (based on the model).

TRM requires only function evaluations, not gradients, to construct and then minimize the model. However, unlike most Bayesian optimization-based approaches, TRM uses a local (often quadratic) model for ℓ around the current iterate, rather than a surrogate that fits all previous points. In taking a step based on this local information, TRM resembles gradient-based approaches.

Following the methods from [27], who assume an additively separable and quadratic upper-level loss function[3], *e.g.*,

$$\ell(\boldsymbol{\gamma}) = \frac{1}{J}\sum_{j=1}^{J} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}_j(\boldsymbol{\gamma})) = \frac{1}{J}\sum_{j=1}^{J} \underbrace{(\hat{\boldsymbol{x}}_j(\boldsymbol{\gamma}) - \boldsymbol{x}_j^{\text{true}})}_{r_j(\boldsymbol{\gamma}\,;\hat{\boldsymbol{x}}_j(\boldsymbol{\gamma}))}{}^2,$$

[3]One could generalize the method to non-quadratic loss functions by approximating ℓ with its second order Taylor expansion.

an outline for a TRM is

1. Create a quadratic model for the upper-level loss function.
   (a) Select a set of upper-level interpolating points and (approximately) evaluate r at each one. After an initialization, one can generally reuse samples from previous iterations. Ref. [27] discusses requirements on the interpolation set to guarantee a good geometry and conditions for re-setting the interpolation sample.
   (b) Estimate the gradients of r_j by interpolating a set of R samples (recall $\boldsymbol{\gamma} \in \mathbb{F}^R$) of the upper-level loss function. This requires solving a set of R linear equations in R unknowns.
   (c) Model the upper-level by replacing r_j with its tangent-plane approximation: $r_j(\boldsymbol{\gamma} + \delta) \approx r(\boldsymbol{\gamma}) + (\tilde{\nabla} r_j(\boldsymbol{\gamma}))'\delta$, where $\tilde{\nabla} r_j(\boldsymbol{\gamma})$ is the estimated gradient from the previous step.
2. Minimize the model within some trust region to find the next candidate set of upper-level parameters. By construction, this is a simple convex-constrained quadratic problem.
3. Accept or reject the updated parameters and update the trust region. If the ratio between the actual reduction and predicted reduction is low, the model may no longer be a good fit, the update is rejected, and the trust region shrinks.

Recall that evaluating ℓ is typically expensive in bilevel problems as each upper-level function evaluation involves optimizing the lower-level cost. Thus, even constructing the model for a TRM can be expensive. To mitigate this computational complexity, [27] incorporated a dynamic accuracy component, with the accuracy for the lower-level cost initially set relatively loose (leading to rough estimates of ℓ) but increasing with the upper-level iterations (leading to refined estimates of ℓ as the algorithm nears a stationary point).

A main result from [27] is a bound on the number of iterations to reach an ϵ-optimal point (defined as $\min_u \|\nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}^{(u)})\| < \epsilon$, where u indexes the upper-level iterates). The bound derivation assumes (i) Φ

is differentiable in $\boldsymbol{x}$, (ii) Φ is μ-strongly convex, *i.e.*, $\Phi(\boldsymbol{x}) - \frac{\mu}{2}\|\boldsymbol{x}\|^2$ is convex for $\mu > 0$, (iii) the derivative of Φ is Lipschitz continuous, and (iv) the first and second derivative of the lower-level cost with respect to $\boldsymbol{x}$ exist and are continuous. These requirements are satisfied by the example filter learning problem (Ex), when $\boldsymbol{A}$ has full column rank, and more generally when there are certain constraints on the hyperparameters. The iteration bound is a function of the following:

- the tolerance ϵ,
- the trust region parameters (parameters that control the increase and decrease in trust region size based on the actual to predicted reduction, the starting trust region size, and the minimum possible trust region size),
- the initialization for $\boldsymbol{\gamma}$, and
- the maximum possible error between the gradient of the upper-level loss function and the gradient of the model for the upper-level loss within a trust region (when the gradient of ℓ is Lipschitz continuous, this bound is the corresponding Lipschitz constant).

The number of iterations required to reach such an ϵ-optimal point is $\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$ [27] and the number of required upper-level loss function evaluations depends more than linearly on R [109]. The growth with the number of hyperparameters impedes its use in problems with many hyperparameters. However, new techniques such as [110] may be able to decrease or remove the dependency, making TRMs promising alternatives to the gradient-based bilevel methods described in the remainder of this review.

3.3 Summary

Turning from the discussion of the lower-level problem in Section 2, this section concentrated on the other two aspects of bilevel problems: the upper-level loss function and the optimization strategy.

The loss function defines what a "good" hyperparameter is, typically using a metric of image quality to compare $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ to a clean, training image, $\boldsymbol{x}^{\text{true}}$. Variations on squared error are the most common upper-level loss functions. It is well known from statistical estimation that the estimator that minimizes MSE is the conditional mean, $\hat{\boldsymbol{x}}(\boldsymbol{y}) = \mathbb{E}\left[\boldsymbol{x}|\boldsymbol{y}\right]$.

Thus, if MSE is the true metric of interest, then lower-level problems should be designed to try to approximate $\mathbb{E}[\boldsymbol{x}|\boldsymbol{y}]$ closely. Yet lower-level formulations in most bilevel papers are not described as conditional mean estimators or approximations thereof. Section 3.1 discussed many other full reference and no reference options, including ones motivated by human judgements of perceptual quality, from the image quality assessment literature; Section 6.2 gives examples of bilevel methods that use some of these other loss functions.

The second half of this section concentrated on model-free and model-based hyperparameter search strategies. The grid search, CMA-ES, and trust region methods described above all scale at least linearly with the number of hyperparameters. Similarly, Bayesian optimization is best-suited for small hyperparameter dimensions; [107] suggests it is typically used for problems with 20 or fewer hyperparameters.

The remainder of this review considers gradient-based strategies for hyperparameter optimization. The main benefit of gradient-based methods is that they can scale to the large number of hyperparameters that are commonly used in machine learning applications. Correspondingly, the main drawbacks of a gradient-based method over the methods discussed in this section are the implementation complexity, the per-iteration computational complexity, and the differentiability requirement. Sections 4 and 5 discuss multiple options for gradient-based methods.

4

Gradient Based Bilevel Methodology: The Groundwork

When the lower-level optimization problem (LL) has a closed-form solution, $\hat{\boldsymbol{x}}$, one can substitute that solution into the upper-level loss function (UL). In this case, the bilevel problem is equivalent to a single level problem and one can use classic single-level optimization methods to minimize the upper-level loss. (See [60] for analysis and discussion of some simple bilevel problems with closed-form solutions for $\hat{\boldsymbol{x}}$.) This review focuses on the more typical bilevel problems that lack a closed-form solution for $\hat{\boldsymbol{x}}$.

Although there are a wide variety of optimization methods for this challenging category of bilevel problems, many methods are built on gradient descent of the upper-level loss. The primary challenge with gradient-based methods is that the gradient of the upper-level function depends on a variable that is itself the solution to an optimization problem involving the hyperparameters of interest. This section describes two common approaches for overcoming this challenge. The first approach uses the fact that the gradient of the lower-level cost function is zero at the minimizer to compute an exact gradient at the exact minimizer. The second approach uses knowledge of the update scheme for the lower-level cost function to calculate the exact gradient for an

approximation to the minimizer after a specific number of lower-level optimization steps.

With this (approximation of the) gradient of the lower-level optimization variable with respect to the hyperparameters, one can compute the gradient of the upper-level loss function with respect to the hyperparameters, $\boldsymbol{\gamma}$. Section 5 uses the building blocks from this section to explain various bilevel methods based on this gradient.

4.1 Set-up

Recall from Section 1.2 that a generic bilevel problem is

$$\operatorname*{argmin}_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) \text{ where } \hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \operatorname*{argmin}_{\boldsymbol{x}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}). \tag{4.1}$$

For simplicity, hereafter we focus on the case $\mathbb{F} = \mathbb{R}$. Using the chain rule, the gradient of the upper-level loss function with respect to the hyperparameters is

$$\nabla \ell(\boldsymbol{\gamma}) = \nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) + (\nabla_{\boldsymbol{\gamma}} \hat{\boldsymbol{x}}(\boldsymbol{\gamma}))' \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})), \tag{4.2}$$

where on the right hand side $\nabla_{\boldsymbol{\gamma}}$ and $\nabla_{\boldsymbol{x}}$ denote partial derivatives w.r.t. the first and second arguments of $\ell(\boldsymbol{\gamma}\,; \boldsymbol{x})$, respectively. We typically select the loss function such that it is easy to compute these partials. For example, if ℓ is the squared error training loss, *i.e.*, $\ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) = \frac{1}{2} \left\| \hat{\boldsymbol{x}}(\boldsymbol{\gamma}) - \boldsymbol{x}^{\text{true}} \right\|_2^2$, then

$$\nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) = 0 \text{ and } \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) = \hat{\boldsymbol{x}}(\boldsymbol{\gamma}) - \boldsymbol{x}^{\text{true}}.$$

The following sections survey methods to find the remaining, more challenging piece in (4.2): the Jacobian $\nabla_{\boldsymbol{\gamma}} \hat{\boldsymbol{x}}(\boldsymbol{\gamma}) \in \mathbb{F}^{N \times R}$ for a given value of $\boldsymbol{\gamma}$.

4.2 Minimizer Approach

The first approach finds the Jacobian $\nabla_{\boldsymbol{\gamma}} \hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ by assuming the gradient of Φ at the minimizer is zero. There are two ways to arrive at the final expression: the implicit function theorem (IFT) perspective (as in [31], [111]) and the Lagrangian/KKT transformation perspective (as

in [30], [32]). This section presents both perspectives in sequence. The end of the section summarizes the required assumptions and discusses computational complexity and memory requirements.

The first step in both perspectives is to assume we have computed $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ and that the lower-level problem 4.1 is unconstrained (*e.g.*, no non-negativity or box constraints). Therefore, the gradient of Φ with respect to $\boldsymbol{x}$ and evaluated at $\hat{\boldsymbol{x}}$ must be zero:

$$\nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})\Big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}(\boldsymbol{\gamma})} = \nabla_{\boldsymbol{x}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}) = \mathbf{0}. \tag{4.3}$$

After this point, the two perspectives diverge.

4.2.1 Implicit Function Theorem Perspective

In the IFT perspective, we apply the IFT (*cf*. [112]) to define a function h such that $\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = h(\boldsymbol{y},\boldsymbol{\gamma})$. If we could write h explicitly, then the bilevel problem could be converted to an equivalent single-level. However, per the IFT, we do not need to define h, we only state that such an h exists. Combining this definition with (4.3) yields

$$\mathbf{0} = \nabla_{\boldsymbol{x}}\Phi(h(\boldsymbol{y},\boldsymbol{\gamma});\boldsymbol{\gamma}). \tag{4.4}$$

Using the chain rule, we differentiate both sides of (4.4) with respect to $\boldsymbol{\gamma}$. The $\boldsymbol{I}$ in the equation below follows from the chain rule because $\nabla_{\boldsymbol{\gamma}}\boldsymbol{\gamma} = \boldsymbol{I}$. We then rearrange terms to solve for the desired quantity, noting that $\nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \nabla_{\boldsymbol{\gamma}}h(\boldsymbol{y},\boldsymbol{\gamma})$. Thus, evaluating all terms at $\hat{\boldsymbol{x}}$ leads to the Jacobian expression of interest:

$$\begin{aligned} 0 =& \nabla_{\boldsymbol{xx}}\Phi(h(\boldsymbol{y},\boldsymbol{\gamma});\boldsymbol{\gamma})\nabla_{\boldsymbol{\gamma}}h(\boldsymbol{y},\boldsymbol{\gamma}) + \boldsymbol{I}\cdot\nabla_{\boldsymbol{x\gamma}}\Phi(h(\boldsymbol{y},\boldsymbol{\gamma});\boldsymbol{\gamma}) \\ \nabla_{\boldsymbol{\gamma}}h(\boldsymbol{y},\boldsymbol{\gamma}) =& -\left[\nabla_{\boldsymbol{xx}}\Phi(h(\boldsymbol{y},\boldsymbol{\gamma});\boldsymbol{\gamma})\right]^{-1}\cdot\nabla_{\boldsymbol{x\gamma}}\Phi(h(\boldsymbol{y},\boldsymbol{\gamma});\boldsymbol{\gamma}) \\ \nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) =& -\left[\nabla_{\boldsymbol{xx}}\Phi(\hat{\boldsymbol{x}};\boldsymbol{\gamma})\right]^{-1}\cdot\nabla_{\boldsymbol{x\gamma}}\Phi(\hat{\boldsymbol{x}};\boldsymbol{\gamma}). \end{aligned} \tag{4.5}$$

When Φ is strictly convex, the Hessian of Φ is positive definite and $\nabla_{\boldsymbol{xx}}\Phi(\hat{\boldsymbol{x}};\boldsymbol{\gamma})$ is invertible.

Substituting (4.5) into (4.2) yields the following expression for the gradient of the upper-level loss function with respect to $\boldsymbol{\gamma}$:

$$\nabla\ell(\boldsymbol{\gamma}) = \nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\hat{\boldsymbol{x}}(\boldsymbol{\gamma})) - \left(\nabla_{\boldsymbol{x\gamma}}\Phi(\hat{\boldsymbol{x}};\boldsymbol{\gamma})\right)'\left(\nabla_{\boldsymbol{xx}}\Phi\left(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}\right)\right)^{-1}\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\hat{\boldsymbol{x}}).$$

If there is a closed-form solution to the lower-level problem, one can verify that the IFT gradient agrees with the analytic gradient; see [111] for examples.

4.2.2 KKT Conditions

In the Lagrangian perspective, (4.3) is treated as a constraint on the upper-level problem, creating a single-level problem with N equality constraints:

$$\underset{\boldsymbol{\gamma}}{\operatorname{argmin}}\, \ell(\boldsymbol{\gamma}\,;\boldsymbol{x}) \text{ subject to } \nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}) = \mathbf{0}_N. \tag{4.6}$$

Using the KKT conditions to transform the bilevel problem into a single-level, constrained problem is sometimes called the "KKT transformation" of the bilevel problem. This transformation relates bilevel optimization to mathematical programs with equilibrium constraints (MPEC); see [9, Ch. 12] and some authors use approaches from the broader MPEC literature to approach bilevel problems [113]. The Lagrangian corresponding to (4.6) is

$$L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) = \ell(\boldsymbol{\gamma}\,;\boldsymbol{x}) + \boldsymbol{\nu}^T \nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$$

where $\boldsymbol{\nu} \in \mathbb{F}^N$ is a vector of Lagrange multipliers associated with the N equality constraints in (4.6).

The Lagrange reformulation is generally well-posed because many bilevel problems, such as (Ex), satisfy the linear independence constraint qualification (LICQ) [8], [114]. The LICQ requires that the matrix of derivatives of the constraint has full row rank [114], *i.e.*,

$$\operatorname{rank}\left(\begin{bmatrix}\nabla_{\boldsymbol{x}\boldsymbol{\gamma}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}) & \nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})\end{bmatrix}\right) = N.$$

Strict convexity of $\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ is therefore a sufficient condition for LICQ to hold. (Note the similarity to the IFT perspective, where strict convexity is sufficient for the Hessian to be invertible.) Ref. [115] explores more generally how bilevel problems relate to MPECs and when the global and local minimizers of the KKT reformulation are minimizers of the original bilevel problem.

The first KKT condition states that, at the optimal point, the gradient of the Lagrangian with respect to $\boldsymbol{x}$ must be $\mathbf{0}$. We can use

this fact to solve for the optimal Lagrangian multiplier, $\hat{\boldsymbol{\nu}}$:

$$\nabla_{\boldsymbol{x}} L(\hat{\boldsymbol{x}}, \boldsymbol{\gamma}, \hat{\boldsymbol{\nu}}) = \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}) + \nabla_{\boldsymbol{x}\boldsymbol{x}} \Phi(\hat{\boldsymbol{x}}\,; \boldsymbol{\gamma}) \hat{\boldsymbol{\nu}} = \mathbf{0}$$
$$\hat{\boldsymbol{\nu}} = -(\nabla_{\boldsymbol{x}\boldsymbol{x}} \Phi\,(\hat{\boldsymbol{x}}\,; \boldsymbol{\gamma}))^{-1} \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}).$$

Substituting the expression for $\hat{\boldsymbol{\nu}}$ into the gradient of the Lagrangian with respect to $\boldsymbol{\gamma}$ yields

$$\begin{aligned}\nabla_{\boldsymbol{\gamma}} L(\hat{\boldsymbol{x}}, \boldsymbol{\gamma}, \hat{\boldsymbol{\nu}}) &= \nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}) + (\nabla_{\boldsymbol{x}\boldsymbol{\gamma}} \Phi(\hat{\boldsymbol{x}}\,; \boldsymbol{\gamma}))' \hat{\boldsymbol{\nu}} \\ &= \nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}) - (\nabla_{\boldsymbol{x}\boldsymbol{\gamma}} \Phi(\hat{\boldsymbol{x}}\,; \boldsymbol{\gamma}))' (\nabla_{\boldsymbol{x}\boldsymbol{x}} \Phi\,(\hat{\boldsymbol{x}}\,; \boldsymbol{\gamma}))^{-1} \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}),\end{aligned}$$

which is equivalent to (4.8).

Ref. [32] generalized the Lagrangian approach to the case where the forward model is defined only implicitly, *e.g.*, as the solution to a differential equation. The authors write the lower-level problem as

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \min_{\tilde{\boldsymbol{y}}} \|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|_2^2 + R(\boldsymbol{x}) \text{ s.t. } e(\tilde{\boldsymbol{y}}, \boldsymbol{x}) = 0, \tag{4.7}$$

where the constraint function, e, incorporates the implicit system model. For example, when the forward model is linear ($\boldsymbol{A}\boldsymbol{x}$), taking $e(\tilde{\boldsymbol{y}}, \boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \tilde{\boldsymbol{y}}\|_2^2$ shows the equivalence of the approach here to the one in [32].

4.2.3 Summary of Minimizer Approach

In summary, the upper-level gradient expression for the minimizer approach (*i.e.*, when one "exactly" minimizes the lower-level cost function) is

$$\nabla \ell(\boldsymbol{\gamma}) = \nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}) - (\nabla_{\boldsymbol{x}\boldsymbol{\gamma}} \Phi(\hat{\boldsymbol{x}}; \boldsymbol{\gamma}))' (\nabla_{\boldsymbol{x}\boldsymbol{x}} \Phi\,(\hat{\boldsymbol{x}}\,; \boldsymbol{\gamma}))^{-1} \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}). \tag{4.8}$$

Thus, for a given loss function and cost function, calculating the gradient of the upper-level loss function (with respect to $\boldsymbol{\gamma}$) requires the following components all evaluated at $\boldsymbol{x} = \hat{\boldsymbol{x}}$: $\nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}) \in \mathbb{F}^R$, $\nabla_{\boldsymbol{x}\boldsymbol{\gamma}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}) \in \mathbb{F}^{N \times R}$, $\nabla_{\boldsymbol{x}\boldsymbol{x}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}) \in \mathbb{F}^{N \times N}$, and $\nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}) \in \mathbb{F}^N$.

Continuing the specific example of learning filter coefficients and

tuning parameters (Ex), the components are:

$$\begin{aligned}
\nabla_{\boldsymbol{x}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}) &= \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}) + e^{\beta_0}\sum_{k=1}^{K} e^{\beta_k}\tilde{\boldsymbol{c}}_k \circledast \dot{\phi}.(\boldsymbol{c}_k \circledast \boldsymbol{x};\epsilon)\\
\nabla_{\boldsymbol{x}\beta_k}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}) &= e^{\beta_0+\beta_k}\tilde{\boldsymbol{c}}_k \circledast \dot{\phi}.(\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}})\\
\nabla_{\boldsymbol{x}c_{k,s}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}) &= e^{\beta_0+\beta_k}\left(\dot{\phi}.((\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}})^{\langle \boldsymbol{s}\rangle}) + \tilde{\boldsymbol{c}}_k \circledast \left(\ddot{\phi}.(\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}}) \odot \hat{\boldsymbol{x}}^{\langle -\boldsymbol{s}\rangle}\right)\right)\\
\nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}) &= \boldsymbol{A}'\boldsymbol{A} + e^{\beta_0}\sum_{k} e^{\beta_k}\boldsymbol{C}_k'\mathrm{diag}(\ddot{\phi}.(\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}}))\boldsymbol{C}_k\\
\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}) &= 0\\
\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\hat{\boldsymbol{x}}) &= \hat{\boldsymbol{x}}(\boldsymbol{\gamma}) - \boldsymbol{x}^{\mathrm{true}}.
\end{aligned} \tag{4.9}$$

Here, the notation $\boldsymbol{x}^{\langle \boldsymbol{i}\rangle}$ means circularly shifting the vector $\boldsymbol{x}$ by $\boldsymbol{i}$ elements, and $c_{k,\boldsymbol{s}}$ denotes the $\boldsymbol{s}$th element of the kth filter $\boldsymbol{c}_k$, where $\boldsymbol{s}$ is a tuple that indexes each dimension of $\boldsymbol{c}_k$. Appendix C.1 gives examples of using the $\boldsymbol{x}^{\langle \boldsymbol{i}\rangle}$ notation and derives $\nabla_{c_{k,s}}(\tilde{\boldsymbol{c}}_k \circledast f.(\boldsymbol{c}_k \circledast \boldsymbol{x}))$, which is the key step to expressing $\nabla_{\boldsymbol{x}c_{k,\boldsymbol{s}}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma})$. The other components follow directly from $\nabla_{\boldsymbol{x}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma})$ using standard gradient tools for matrix expressions [116].

The minimizer approach to finding $\nabla\ell(\boldsymbol{\gamma})$ uses the following assumptions:

1. Both the upper and lower optimization problems have no inequality constraints.
2. $\hat{\boldsymbol{x}}$ is the minimizer to the lower-level cost function, not an approximation of the minimizer. This constraint ensures that (4.3) holds.
3. The cost function Φ is twice-differentiable in $\boldsymbol{x}$ and differentiable with respect to $\boldsymbol{x}$ and $\boldsymbol{\gamma}$.
4. The Hessian of the lower-level cost function, $\nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$, is invertible; this is guaranteed when Φ is strictly convex.

The first condition technically excludes applications like CT imaging, where the image is typically constrained to be non-negative. However, non-negativity constraints are rarely required when good regularizers are used, so the resulting non-constrained image can still be useful in practice [112].

The second constraint is often the most challenging since the lower-level problem typically uses an iterative algorithm that runs for a certain number of iterations or until a given convergence criteria is met. As previously noted, if there were a closed-form solution for $\hat{\boldsymbol{x}}$, then we would not have needed to use the IFT or Lagrangian to find the partial derivative of $\hat{\boldsymbol{x}}$ with respect to $\boldsymbol{\gamma}$. Since one usually does not reach the exact minimizer, the calculated gradient will have some error in it, depending on how close the final iterate is to the true minimizer $\hat{\boldsymbol{x}}$. Thus, the practical application of this method is more accurately called Approximate Implicit Differentiation (AID) [117], [118]. Section 4.5 further discusses gradient accuracy.

The third condition disqualifies sparsity-promoting functions such as the 0-norm and 1-norm as choices for ϕ.

Finally, the fourth (strict convexity) condition is easily satisfied in denoising problems where $\boldsymbol{A} = \boldsymbol{I}$ whenever ϕ is convex. Common convex ϕ choices include (CR1N) and the Fair potential [119]. However, in applications like compressed sensing where $\boldsymbol{A}'\boldsymbol{A}$ is not positive definite, the strict convexity of Φ depends non-trivially on $\boldsymbol{\gamma}$. The condition is likely to hold in practice for "good" values of $\boldsymbol{\gamma}$. Specifically, if ϕ is strictly convex, then the condition will hold for any value of $\boldsymbol{\gamma}$ such that the null-space of the regularization term is disjoint from the null-space of $\boldsymbol{A}$ and the regularization parameters are sufficiently large (e^{β_k} cannot approach 0). To interpret this condition, recall that regularization helps compensate for the under-determined nature of $\boldsymbol{A}$ (Section 2.1). Values of $\boldsymbol{\gamma}$ that do not sufficiently "fill-in" the null-space of $\boldsymbol{A}$ will leave the lower-level cost function under-determined. The task-based nature of the bilevel problem should discourage these "bad" values, but this intuition is insufficient to claim that the minimizer approach is well-defined at all iterations. To ensure that the lower-level problem is strongly convex, one could include a term like $\|\boldsymbol{x}\|_2^2$ with a small positive regularization parameter, like is done with elastic-net regularization [120].

4.2.4 Computational Costs

The largest cost in computing the gradient of the upper-level loss using (4.8) is often finding (an approximation of) $\hat{\boldsymbol{x}}$. However, this cost is

difficult to quantify, as the IFT approach is agnostic to the lower-level optimization methodology. To compare the bilevel gradient methods, we will later assume the cost is comparable to the gradient descent calculations used in the unrolled approach (described in Section 4.4). However, this is an over-estimation of the cost, as the IFT approach is not constrained to smooth lower-level updates, and one can use optimization methods with, *e.g.*, warm starts and restarts to reduce this cost.

When the lower-level problem satisfies the assumptions above, and assuming one has already found $\hat{\boldsymbol{x}}$, a straight-forward approach to computing the gradient (4.8) would be dominated by the $\mathcal{O}(N^3)$ operations required to compute the Hessian's inverse. For many problems, N is large, and that matrix inversion is infeasible due to computation or memory requirements. Instead, as described in [121], one can use a conjugate gradient (CG) method to compute the matrix-vector product

$$\left(\nabla_{\boldsymbol{xx}} \Phi\left(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}\right)\right)^{-1} \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,;\hat{\boldsymbol{x}}) \tag{4.10}$$

because the Hessian is symmetric and positive definite (see assumption #4 in the previous section). For a generic $\boldsymbol{A}$, each CG iteration requires multiplying the Hessian by a vector, which is $\mathcal{O}(N^2)$.

CG takes N iterations to converge fully (ignoring finite numerical precision), so the final complexity is still $\mathcal{O}(N^3)$ in general. However, the Hessian often has a special structure that simplifies computing the matrix-vector product. Consider the running example of learning filters per (Ex). The Hessian, as given in (4.9), multiplied with any vector $\boldsymbol{v} \in \mathbb{F}^N$ is

$$\nabla_{\boldsymbol{xx}} \Phi(\hat{\boldsymbol{x}}; \boldsymbol{\gamma}, \boldsymbol{y}) \cdot \boldsymbol{v} =$$

$$\underbrace{\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{v})}_{2N^2} + e^{\beta_0} \sum_k e^{\beta_k} \underbrace{\boldsymbol{C}_k'}_{NS} \cdot \overbrace{\operatorname{diag}(\ddot{\phi}.(\underbrace{\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}}}_{NS}))}^{N} \cdot \underbrace{(\boldsymbol{C}_k \boldsymbol{v})}_{NS}. \tag{4.11}$$

The annotations show the multiplications required for each component, where we used the simplifying assumption that the number of measurements matches the number of unknowns ($M = N$).

As written, (4.11) does not made any assumptions on $\boldsymbol{A}$, so the first term is still computationally expensive. If $\boldsymbol{A}$ is the identity matrix

(as in denoising), the N^2 term could instead be zero cost. If $\boldsymbol{A}'\boldsymbol{A}$ is circulant, *e.g.*, if $\boldsymbol{A}$ is a MRI sampling matrix that can be written in terms of a discrete Fourier transform, then the cost is $N\log(N)$. More generally, the computational cost for one (of N) iterations of CG is $\mathcal{O}(c_{\boldsymbol{A}}N)$ where $c_{\boldsymbol{A}} \in [0, N]$ is some constant dependent on the structure of $\boldsymbol{A}$.

For the second addend in (4.11), we assume that $S \ll N$, so direct convolution is most efficient and the matrix-vector product requires $\mathcal{O}(NS)$ multiplies. When the filters are relatively large, one can use Fourier transforms for the filtering, and the cost is $\mathcal{O}(N\log(N))$. The final cost of the Hessian-vector product for (Ex) is $\mathcal{O}(c_{\boldsymbol{A}}N + RN)$. This cost includes a multiplication by K to account for the sum over all filters, which simplifies since SK is[1] $\mathcal{O}(R)$.

If N is small enough that storing the inverse Hessian is feasible, then one can estimate the Hessian inverse rather than computing it directly. Consider using a quasi-Newton algorithm to find $\hat{\boldsymbol{x}}$, which involves estimating the inverse Hessian as a pre-conditioning matrix for the gradient steps. This inverse Hessian estimate can be "shared" to efficiently approximate the inverse Hessian-vector product in (4.8) [89]. Ref. [122] used this strategy and also incorporated information from the upper-level loss function to improve the estimated inverse Hessian vector product while maintaining the super-linear convergence rate of the quasi-Newton algorithm.

4.3 Translation to a Single Level

Before discussing the other widely used approach to calculating the gradient of the upper-level loss, we summarize a specialized approach for 1-norm regularizers. Like the minimizer approach described above, this approach assumes we have computed an (almost) exact minimizer of the lower-level cost function. It writes the minimizer as an (almost everywhere) differentiable function in terms of that $\hat{\boldsymbol{x}}$, then substitutes this expression for the minimizer into the upper-level loss to create a

[1] The full parameter dimension includes the filters and tuning parameters, so $R = S(K + 1) + 1$.

single-level optimization problem that is suitable for one hyperparameter update step.

Ref. [123] proposed the translation to a single-level approach to solve a bilevel problem with both synthesis and analysis operators. Refs. [124], [125] more recently presented versions specific to analysis operators. The bilevel problem considered in [124], [125] is:

$$\begin{aligned} \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \sum_j \frac{1}{2} \|\hat{\boldsymbol{x}}_j(\boldsymbol{\gamma}) - \boldsymbol{x}_j^{\text{true}}\|_2^2 \\ \hat{\boldsymbol{x}}_j(\boldsymbol{\gamma}) = \underset{\boldsymbol{x}\in\mathbb{F}^N}{\operatorname{argmin}} \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{y}_j\|_2^2 + \|\boldsymbol{\Omega}_{\boldsymbol{\gamma}} \boldsymbol{x}\|_1 , \end{aligned} \tag{4.12}$$

where $\boldsymbol{\Omega}_{\boldsymbol{\gamma}} \in \mathbb{F}^{F\times N}$ is a matrix constructed based on $\boldsymbol{\gamma}$. We write $\boldsymbol{\Omega}$ without the $\boldsymbol{\gamma}$ subscript and $\hat{\boldsymbol{x}}_j(\boldsymbol{\gamma})$ without the j subscript in the following discussion to simplify notation. As in the minimizer approach, the first step is to compute $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ for the current guess of $\boldsymbol{\gamma}$, *e.g.*, using ADMM. After optimizing for $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$, [124], [125] both used the known sign pattern of the filtered signal, $\boldsymbol{\Omega}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ to rewrite the lower-level problem (4.12) in a simpler, (almost everywhere) differentiable form. By rewriting the problem, the translation to a single level approaches handle the non-smooth 1-norm in (4.12) directly–they do not require any corner rounding as in the minimizer approach.

One way to rewrite the lower-level problem is to split the 1-norm into its positive and negative elements, *e.g.*,

$$\|\boldsymbol{\Omega}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\|_1 = \sum_{i\in\mathcal{I}_+(\boldsymbol{\gamma})} [\boldsymbol{\Omega}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})]_i - \sum_{i\in\mathcal{I}_-(\boldsymbol{\gamma})} [\boldsymbol{\Omega}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})]_i,$$

where $\mathcal{I}_+(\boldsymbol{\gamma})$ and $\mathcal{I}_-(\boldsymbol{\gamma})$ denote the set of indices where $\boldsymbol{\Omega}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ is positive and negative, respectively. Ref. [124] used this approach and defined a diagonal sign matrix, $\boldsymbol{S}(\boldsymbol{\gamma}) = \operatorname{diag}(\operatorname{sign}(\boldsymbol{\Omega}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})))$, having positive and negative diagonal elements at the appropriate indices. For a single training image, the lower-level problem (4.12) is thus equivalent to

$$\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \underset{\boldsymbol{x}\in\mathbb{F}^N}{\operatorname{argmin}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \mathbf{1}' \boldsymbol{S}(\boldsymbol{\gamma})\boldsymbol{\Omega}\boldsymbol{x}, \text{ s.t. } [\boldsymbol{\Omega}\boldsymbol{x}]_{\mathcal{I}_0(\boldsymbol{\gamma})} = \mathbf{0}, \tag{4.13}$$

where $\mathcal{I}_0(\boldsymbol{\gamma})$ denotes the set of indices where $[\boldsymbol{\Omega}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})]_i = 0$. The rewritten problem (4.13) it is a quadratic cost function with a linear equality

constraint and thus has a closed-form solution. Ref. [124] states that $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ is differentiable everywhere except a set of measure zero when $\boldsymbol{A} = \boldsymbol{I}$ and when the rows of $\boldsymbol{\Omega}$ corresponding to $\mathcal{I}_0(\boldsymbol{\gamma})$ are linearly independent.

Another way to rewrite (4.12) uses the results from [126]. The lower-level problem (4.12) can be transformed into the dual problem

$$\min_{\boldsymbol{d} \in \mathbb{R}^F} \frac{1}{2} \left\| \text{-}\boldsymbol{\Omega}'\boldsymbol{d} + \boldsymbol{y} \right\|^2 - \frac{1}{2} \left\| \boldsymbol{y} \right\|^2 \text{ s.t. } |d_i| \leq 1 \; \forall i. \tag{4.14}$$

where the dual variable $\boldsymbol{d}$ is related to the filtered signal by

$$\boldsymbol{d}_i \in \begin{cases} \operatorname{sign}([\boldsymbol{\Omega}\boldsymbol{x}]_i) & \text{if } [\boldsymbol{\Omega}\hat{\boldsymbol{x}}]_i \neq 0 \\ [\text{-}1, 1] & \text{if } [\boldsymbol{\Omega}\hat{\boldsymbol{x}}]_i = 0 \end{cases} \tag{4.15}$$

(compare to (A.9) and (A.13) in Appendix A). Ref. [126] defines boundary indices as the set of indices where the dual variable is at the edges of its allowed range: $B := \{i : |\boldsymbol{d}_i| = 1\}$. The complement to this set is $\bar{B} := \{i : |\boldsymbol{d}_i| \neq 1\}$ and contains all coordinates where $\boldsymbol{d}$ is in the interior of its allowed range. Let $\boldsymbol{\Omega}_{\boldsymbol{d}} \in \mathbb{F}^{|B| \times N}$ contain the rows of $\boldsymbol{\Omega}$ that correspond to B and similarly for $\boldsymbol{\Omega}_{\bar{B}}$. By taking the gradient of the Lagrangian of the dual formulation and then substituting the dual variable minimizer into (A.11), [126] derives the following closed-form expression for $\hat{\boldsymbol{x}}$

$$\hat{\boldsymbol{x}} = \left(\boldsymbol{I} - \boldsymbol{\Omega}_{\bar{B}}^{+} \boldsymbol{\Omega}_{\bar{B}}\right) \left(\boldsymbol{y} - \boldsymbol{\Omega}_B \operatorname{sign}(\boldsymbol{\Omega}_B \hat{\boldsymbol{x}})\right), \tag{4.16}$$

which is a projection onto the null space of $\boldsymbol{\Omega}_{\bar{B}}$. Thus, similar to splitting the 1-norm based on the sign of $\boldsymbol{\Omega}\hat{\boldsymbol{x}}$, splitting the dual variable into boundary and interior indices yields a rewritten problem with a simpler structure.

Ref. [125] used (4.16) to rewrite the lower-level problem (4.12) and then used matrix gradient relations to derive a closed-form expression for $\nabla_{\boldsymbol{\gamma}} \hat{\boldsymbol{x}}(\boldsymbol{\gamma})$. Unlike [124], the final upper-level gradient $\nabla \ell(\boldsymbol{\gamma})$ in [125] does not require that the rows of $\boldsymbol{\Omega}$ that are orthogonal to $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ are linearly independent.

In both (4.13) and (4.16), the rewritten problem has the same minimizer as the original problem (4.12), but the reformulated problem has a simpler structure. Recall that the rewriting process requires $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$,

so one cannot use this equivalence to optimize the lower-level problem. However, the closed-form expressions can be differentiated. Because of the discontinuity of the sign function, both methods require the sign pattern of $\boldsymbol{\Omega}\hat{\boldsymbol{x}}$ to be constant within a region to compute an accurate gradient [124], [125]. The authors have shown that this condition holds in various empirical settings [127].

In summary, the translation to a single level approach involves computing $\hat{\boldsymbol{x}}$, creating a closed-form expression for $\hat{\boldsymbol{x}}$, and then differentiating the closed-form expression to compute the desired Jacobian, $\nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$. As in the minimizer approach, $\nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ is related to the upper-level gradient by the chain rule (4.2). In terms of computation, both translation to a single level approaches require optimizing the lower-level cost sufficiently precisely to ensure the sign pattern converges; [125] used thousands of iterations of ADMM. Ref. [125] demonstrates that evaluating the closed-form expression for $\nabla\ell(\boldsymbol{\gamma})$ is faster than using automatic differentiation tools that rely on backpropagation.

4.4 Unrolled Approaches

A popular approach to finding $\nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ is to assume that the lower-level cost function is approximately minimized by applying T iterations of some (sub)differentiable optimization algorithm, where we write the update step at iteration $t \in [1 \dots T]$ as

$$\boldsymbol{x}^{(t)} = \Psi(\boldsymbol{x}^{(t-1)}; \boldsymbol{\gamma}),$$

for some mapping $\Psi : \mathbb{F}^N \mapsto \mathbb{F}^N$ that should have the fixed-point property $\Psi(\hat{\boldsymbol{x}}(\boldsymbol{\gamma}); \boldsymbol{\gamma}) = \hat{\boldsymbol{x}}(\boldsymbol{\gamma})$. For example, GD has $\Psi(\boldsymbol{x}; \boldsymbol{\gamma}) = \boldsymbol{x} - \alpha_\Phi \nabla\Phi(\boldsymbol{x}; \boldsymbol{\gamma})$ for some step size α_Φ. We write the update here only in terms of $\boldsymbol{x}$; the idea easily extends to updates in terms of a state vector that allows one to include momentum terms, weights, and other accessory variables in $\boldsymbol{\gamma}$ [128].

In contrast to the two approaches described above, the "unrolled" approach no longer assumes the solution to the lower-level problem is an exact minimizer. Instead, the unrolled approach reformulates the

bilevel problem (LL) as

$$\underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \underbrace{\ell\left(\boldsymbol{\gamma}\,;\,\boldsymbol{x}^{(T)}(\boldsymbol{\gamma})\right)}_{\ell(\boldsymbol{\gamma})} \text{ s.t.} \tag{4.17}$$
$$\boldsymbol{x}^{(t)}(\boldsymbol{\gamma}) = \Psi(\boldsymbol{x}^{(t-1)}\,;\boldsymbol{\gamma}), \quad \forall t \in [1 \dots T],$$

where $\boldsymbol{x}^{(0)}$ is an initialization, *e.g.*, $\boldsymbol{A}'\boldsymbol{y}$. One can then take the (sub)gradient of a finite number T of iterations of Ψ, hoping that $\boldsymbol{x}^{(T)}$ approximately minimizes the lower-level function Φ.

The chain rule for derivatives is the foundation of the unrolled method. The gradient of interest, $\nabla\ell(\boldsymbol{\gamma})$, depends on the gradient of the optimization algorithm step with respect to $\boldsymbol{x}$ and $\boldsymbol{\gamma}$. For readability, define the following matrices for the tth unrolled iteration

$$\boldsymbol{H}_t \coloneqq \nabla_{\boldsymbol{x}}\Psi\left(\boldsymbol{x}^{(t-1)}\,;\boldsymbol{\gamma}\right) \in \mathbb{F}^{N\times N} \text{ and } \boldsymbol{J}_t \coloneqq \nabla_{\boldsymbol{\gamma}}\Psi\left(\boldsymbol{x}^{(t-1)}\,;\boldsymbol{\gamma}\right) \in \mathbb{F}^{N\times R},$$

for $t \in [1, T]$. We use these letters because, when using gradient descent as the optimization algorithm, $\nabla_{\boldsymbol{x}}\Psi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ is closely related to the Hessian of Φ and $\nabla_{\boldsymbol{\gamma}}\Psi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ is proportional to the Jacobian of the gradient[2]. Thus, when Ψ corresponds to GD, an unrolled approach involves computing the same quantities as required by the IFT approach (4.8).

By the chain rule, the gradient of (4.17) is

$$\nabla\ell(\boldsymbol{\gamma}) = \nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)}) + \left(\sum_{t=1}^{T}\left(\boldsymbol{H}_T\cdots\boldsymbol{H}_{t+1}\right)\boldsymbol{J}_t\right)' \nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)}) \in \mathbb{F}^R. \tag{4.18}$$

One can derive this gradient expression using a reverse or forward perspective, with parallels to back-propagation through time and real-time recurrent learning respectively [128]. Appendix B describes the reverse and forward approaches to unrolling.

Most unrolled implementations use the reverse-mode approach (back-propagation) due to its lower computational burden, but unrolling with reverse mode differentiation may have prohibitively high memory requirements if T is large or if the training dataset includes large images

[2]When $\Psi(\boldsymbol{x}\,;\boldsymbol{\gamma}) = \boldsymbol{x} - \alpha_\Phi\nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$, then $\nabla_{\boldsymbol{x}}\Psi(\boldsymbol{x}\,;\boldsymbol{\gamma}) = \boldsymbol{I} - \alpha_\Phi\nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ and $\nabla_{\boldsymbol{\gamma}}\Psi(\boldsymbol{x}\,;\boldsymbol{\gamma}) = -\alpha_\Phi\nabla_{\boldsymbol{x}\boldsymbol{\gamma}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$.

[68]. A strategy to trade-off the memory and computation requirements is checkpointing, which stores $\boldsymbol{x}$ every few iterations. Checkpointing is an active research area; see [129] for an overview. Another option is to use (some or all) reversible network layers [130] to trade off the memory and computational requirements.

The following sections overview some design decisions for unrolling and draw some parallels to unrolled methods as used in the (non-bilevel specific) machine learning literature. Section 7.1 further discusses the relation between bilevel problems and unrolling methods common in the broader literature.

4.4.1 Number of Iterations

Unlike the minimizer approach, where the goal is to run the lower-level optimization until (close to) convergence so that an optimally condition holds and one can use implicit differentiation to find $\nabla \ell(\boldsymbol{\gamma})$, most unrolling methods set the number of lower-level iterations T in advance. The set number of lower-level iterations mimics the depth of neural networks and allows a precise estimate of how much computational effort each lower-level optimization takes. The chosen number of iterations is important as, at test time, "one cannot deviate from the choice of [number of unrolled iterations] and expect good performance" [131].

Although it is generally not equal to the gradient of the original bilevel problem (UL), the unrolled gradient is exact for the reformulated problem (4.17). Therefore, when T is small enough that the lower-level optimizer is far from convergence, the unrolled method is only loosely tied to the original bilevel optimization problem. To maintain a stronger connection to the bilevel problem while avoiding setting T larger than necessary for convergence, [132] used a convergence criterion to determine the number of Ψ iterations rather than pre-specifying a number of iterations. Unrolling until convergence is also used in deep equilibrium or fixed point networks, see Section 7.2.

A subtle point in unrolling gradient-based methods for the lower-level cost function is that the Lipschitz constant of its gradient is a function of the hyperparameters, so the step size range that ensures convergence cannot be pre-specified. Many unrolled methods use a

fixed step size alongside a fixed T and allow the learned parameters to adapt to these set values. An alternative approach is to compute a new step-size as a function of the current parameters, $\boldsymbol{\gamma}^{(u)}$, every upper-level iteration. For example, from (C.5), for a given value $\boldsymbol{\gamma}$ of the tuning parameters and filter coefficients, a Lipschitz constant of the lower-level gradient for (Ex) is

$$L = \sigma_1^2(\boldsymbol{A}) + e^{\beta_0} L_{\dot{\phi}} \sum_k e^{\beta_k} \|\boldsymbol{c}_k\|_1^2 , \tag{4.19}$$

where $L_{\dot{\phi}}$ is a Lipschitz constant for $\dot{\phi}(z)$ (for (CR1N), $L_{\dot{\phi}} = 1/\epsilon$). A reasonable step size for the classical gradient descent method would be $1/L$. It is relatively inexpensive to update this L as $\boldsymbol{\gamma}$ evolves.

The adaptive approach to setting the step size ensures that any theoretical guarantees of the lower-level optimizer hold. This approach may be beneficial when using a convergence criteria for the lower-level optimization algorithm or when running sufficiently many lower-level iterations to essentially converge. However, updating the step-size every upper-level iteration is incompatible with fixing the number of unrolled iterations. To illustrate, consider an upper-level iteration where the tuning parameters increase, leading to a larger L and a smaller step size. In a fixed number of iterations, the smaller step size means the lower-level optimization algorithm will be farther from convergence, and the estimated minimizer, $\hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(u+1)})$, may be worse (as judged by the upper-level loss function) than $\hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(u)})$, even if the updated hyperparameters are better when evaluated with the previous (larger) step-size or more lower-level iterations.

Another approach is to learn the step-size and/or number of iterations. For example, [25] provides a continuous-time perspective on the unrolling approach and learns the stopping time, which translates to the number of iterations in the discrete approach.

The continuous time perspective on unrolling models the lower-level problem as a differential equation with an initial condition enforcing that $\boldsymbol{x}$ at time 0 is $\boldsymbol{x}_0$ [25], [133]. Just as the unrolled approach better approximates the bilevel problem as the number of iterations approaches infinity, the continuous perspective on unrolling approaches the bilevel problem as the stopping time $T \to \infty$. The discretization of

the continuous-time gradient flow corresponds to an unrolled optimization algorithm (or, more generally, to a variational network with shared weights) and back-propagation can be seen as a discretization of the continuous-time adjoint equation [25], [133]. Solving the differentiable adjoint equation does not require saving the forward-pass output at every "step," making the backward pass feasible for large problems such as 3D CT image reconstruction [134].

Like many other bilevel methods for filter learning, [25] uses a regularizer based on the Field of Experts [55] and the standard data-fit term. The lower-level problem in [25] is

$$\text{State equation: } \frac{d\boldsymbol{x}(t)}{dt} = -\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}(t) - \boldsymbol{y}) - \sum_k \boldsymbol{C}_k' \phi_k(\boldsymbol{C}_k \boldsymbol{x}(t))$$

$$\text{Initial condition: } \boldsymbol{x}(0) = \boldsymbol{x}_0,$$

where [25] learns a separate penalty function for each filter. Ref. [25] found that beyond a certain depth, increasing the number of layers did not significantly decrease the upper-level loss. Further, following intuition, the learned stopping time increased with higher noise levels or blur strengths in the denoising and deblurring problem settings [25].

4.4.2 Application to Non-smooth Cost Functions

An important distinction between the minimizer approach and the unrolled approach is that the unrolled approach depends on the optimization algorithm. Therefore, in addition to the number of iterations and step size, one must select an optimization algorithm to unroll. The choice is typically driven by parameters such as memory availability and desired run-time, with the one requirement being that Ψ be differentiable in both $\boldsymbol{x}$ and $\boldsymbol{\gamma}$. For certain cost functions, a resulting advantage of the unrolling method is that one can use a smooth Ψ to optimize a non-smooth cost function, removing the need for smoothing techniques such as used in (CR1N).

Ochs *et al.* [14] describe one such smooth update algorithm for a non-smooth cost function. At a high-level, their approach is to:

1. transform the lower-level cost function to a primal-dual, saddle-point problem, using the Legendre-Fenchel conjugate of ϕ (defined

in Appendix A),

2. use a forward-backward splitting algorithm to alternatively update the primal ($\boldsymbol{x}$) and dual ($\boldsymbol{d}$) variables, and
3. replace the Euclidean norm in the proximal operator in the dual variable update equation with a Bregman divergence measure.

If the Bregman divergence measure is chosen carefully, the resulting update is smooth and standard backpropagation tools can compute $\nabla \ell(\boldsymbol{\gamma})$. This section overviews how the approach in [14] applies to (Ex). Ref. [14] derives the full backpropagation formula and uses Bregman divergences to unroll non-smooth cost functions in a multi-label segmentation problem, but the approach generalizes to image reconstruction as shown here.

Using the stacked convolutional matrix notation for the learned filters defined in (2.7) and selecting ϕ to be the absolute value function[3], the lower-level optimization problem is

$$\underset{\boldsymbol{x}}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \|\boldsymbol{\Omega}\boldsymbol{x}\|_1 \,.$$

From (A.8), the corresponding saddle-point formulation is

$$\underset{\boldsymbol{x}}{\operatorname{argmin}} \min_{\boldsymbol{d}} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 - \langle \boldsymbol{d}, \boldsymbol{\Omega}\boldsymbol{x}\rangle \text{ s.t. } |d_i| \leq 1 \; \forall i,$$

where $\boldsymbol{d}$ is the dual variable. The minimum cost value and corresponding minimizer, $\hat{\boldsymbol{x}}$, of the saddle-point problem are equivalent to those of the original problem because the 1-norm is convex.

To optimize the saddle-point problem, one can alternate $\boldsymbol{x}$ and $\boldsymbol{z}$ updates. Ref. [14] uses the primal-dual algorithm from [135] that introduces a proximity function to each update step:

$$\begin{aligned} \boldsymbol{x}^{(t+1)} &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 - \langle \boldsymbol{d}^{(t)}, \boldsymbol{\Omega}\boldsymbol{x}\rangle + \frac{1}{\alpha_{\mathrm{x}}}\mathbf{1}' D_{\cdot}(\boldsymbol{x}, \boldsymbol{x}^{(t)}) \\ \boldsymbol{d}^{(t+1)} &= \underset{\boldsymbol{d}}{\operatorname{argmin}} \frac{1}{\alpha_{\mathrm{d}}}\mathbf{1}' D_{\cdot}(\boldsymbol{d}, \tilde{\boldsymbol{d}}) - \langle \boldsymbol{d}, \boldsymbol{\Omega}\tilde{\boldsymbol{x}}\rangle \text{ s.t. } |\boldsymbol{d}_i| \leq 1 \; \forall i, \end{aligned} \tag{4.20}$$

[3] When using the absolute value, one can absorb the tuning parameters β_k into the filter magnitudes, conveniently reducing the dimension of $\boldsymbol{\gamma}$.

where $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{d}}$ are defined in terms of previous iterates, *e.g.*, when including momentum, and α_{x} and α_{d} are step size parameters chosen according to the theory in [135]. The $\boldsymbol{x}$ update is a smooth, quadratic problem and is straight-forward. However, the standard dual update involves a non-smooth projection; in particular, if the proximal distance function is the standard Euclidean 2-norm, i.e., $D(d, \tilde{d}) = \frac{1}{2}(d - \tilde{d})^2$, then the $\boldsymbol{d}$ update is the projection

$$\boldsymbol{d}^{(t+1)} = \text{sign.}(\tilde{\boldsymbol{d}} + \alpha_{\mathrm{d}}\boldsymbol{\Omega}\tilde{\boldsymbol{x}}) \odot \text{min.}(1, |\tilde{\boldsymbol{d}} + \alpha_{\mathrm{d}}\boldsymbol{\Omega}\tilde{\boldsymbol{x}}|),$$

which is non-smooth.

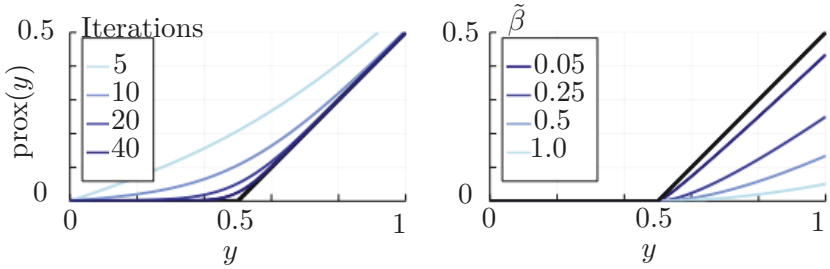
To make the $\boldsymbol{d}$ update smooth, [14] replaces the standard Euclidean norm in the proximity operator with a Bregman divergence. For the 1-norm regularizer, [14] considers the divergence measure

$$D(d, \tilde{d}) = \psi(d) - \psi(\tilde{d}) - \nabla\psi(\tilde{d})'(d - \tilde{d}) \tag{4.21}$$

where $\psi(d) = \dfrac{1}{2}\left((d+1)\log(d+1) + (1-d)\log(1-d)\right)$. Similar to standard distance metrics, this Bregman divergence is zero when $d = \tilde{d}$. However, it is not symmetric, *i.e.*, $D(d, \tilde{d}) \neq D(\tilde{d}, d)$ in general. Using this definition for D, one can differentiate and solve for the minimizer in the $\boldsymbol{d}$ update (4.20) [14]. Because all the functions are separable, the update can be done independently for each $\boldsymbol{d}$ coordinate:

$$d_i^{(t+1)} = \frac{e^{2\alpha_{\mathrm{d}}[\boldsymbol{\Omega}\boldsymbol{x}]_i} - \frac{1-\tilde{d}_i}{1+\tilde{d}_i}}{e^{2\alpha_{\mathrm{d}}[\boldsymbol{\Omega}\boldsymbol{x}]_i} + \frac{1-\tilde{d}_i}{1+\tilde{d}_i}}. \tag{4.22}$$

When the step-size α_{d} approaches infinity, $d_i^{(t+1)}$ approaches ± 1 (its extreme values). When α_{d} approaches 0, $d_i^{(t+1)} = \tilde{d}_i$. The updated coordinate is guaranteed to satisfy the constraint $|d_i| \leq 1$ whenever $\tilde{d}_i$ does, so there is no need for a (non-smooth) projection. Although this approach allows for applying the unrolled method to non-smooth cost functions, [14] comments that "the [equivalent of a] 'smoothing parameter' in our approach is the number of iterations of the algorithm that replaces the lower level problem." Fig. 4.1 demonstrates how the number of iterations impacts the effective smoothing for a simple version of the problem where $\boldsymbol{A} = \boldsymbol{I}$ and $\boldsymbol{\Omega} = \boldsymbol{I}$.



(a) Estimate using the Bregman divergence.

(b) Proximal operator with $p = 3/2$ term.

Figure 4.1: Proximal operators for $R(x) = \frac{1}{2}|x|$ and some smooth relatives. The black line in both plots is the soft thresholding function, which is the proximal operator for the absolute value function, *i.e.*, $\text{prox}(y) = \text{argmin}_x \frac{1}{2}(x-y)^2 + \frac{1}{2}|x|$. (a) As described in [14], the number of iterations of the primal-dual algorithm with the Bregman proximity function acts as a smoothing parameter for the proximal operator estimate and the estimate improves as the number of iterations increases (from light to dark lines). (b) Smooth proximal operator for the non-smooth penalty function (4.23) for $p = 3/2$, $\beta = 0.5$, and four different values of $\tilde{\beta}$. The proximal operator is closer to soft thresholding for smaller values of $\tilde{\beta}$ (darker lines).

Ref. [68] uses the same saddle-point problem as in [14] to propose another approach to computing $\nabla \ell(\boldsymbol{\gamma})$. Instead of unrolling an algorithm and then back-propagating, [68] uses a sensitivity analysis and introduces additional adjoint variables that allow for simultaneously computing $\nabla \ell(\boldsymbol{\gamma})$ in the same forward iteration as $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$, without incurring the large matrix-matrix multiplications costs as in the forward-mode method of computing (4.18). Although the theoretical analysis of the resulting "piggy-backing" optimization algorithm is for smooth functions, [68] found it worked well empirically in non-smooth settings.

Christof [136] shows another approach to achieving a smooth optimization algorithm for a non-smooth cost function. Ref. [136] specifically considers cost functions with penalty functions of the form

$$\phi(z) = \beta|z| + 2\tilde{\beta}\frac{|z|^p}{p} \text{ for } 1 < p < 2. \tag{4.23}$$

As a simple demonstration, in the case where there are no convolutional

filters and $p = 3/2$, the lower-level cost function is the proximal operator

$$\text{prox}_{\phi}(y) = \underset{x}{\text{argmin}} \frac{1}{2}(x - y)^2 + \phi(x).$$

Differentiating and solving for the minimizer yields

$$\text{prox}_{\phi}(y) = \begin{cases} \text{sign}(y) \left(\sqrt{\tilde{\beta}^2 + |y| - \beta} - \tilde{\beta} \right)^2 & \text{if } |y| > \beta \\ 0 & \text{else,} \end{cases}$$

which is continuous and differentiable everywhere with respect to y despite the non-differential absolute value function in ϕ! Fig. 4.1 shows this proximal operator alongside the proximal operator when $\phi(z) = |z|$ (soft thresholding). Ref. [136] proves that this simple example generalizes to the bilevel problem of learning filters.

4.5 Summary

This section focused on computing $\nabla \ell(\boldsymbol{\gamma})$, the gradient of the upper-level loss function with respect to the learnable parameters. Section 5 builds on this foundation to consider optimization methods for bilevel problems. Many of those optimization methods can be used in conjunction with the minimizer, translation to a single level, or unrolled approaches to compute $\nabla \ell(\boldsymbol{\gamma})$. Thus, how one selects an approach may depend on the structure of the specific bilevel problem, how closely tied one wishes to be to the original bilevel problem, computational cost, and/or gradient accuracy.

The translation to a single level approach is tailored to a specific type of bilevel problem. A benefit of the translation approach is the ability to use the 1-norm (without any corner rounding) in the lower-level cost function. However, the corresponding drawback is the (current) lack of generality in the minimizer approach; the closed-form expression derived in [123]–[125] is specific to using the 1-norm as ϕ. Expanding this approach to regularizers other than the 1-norm is a possible avenue for future work.

One difference among the methods is whether they depend on the lower-level optimization algorithm; while the unrolled approach depends

on the specific optimization algorithm, the minimizer approach and the translation to a single level approach do not. A resulting downside of unrolling is that one cannot use techniques such as warm starts and non-differentiable restarts, so $\boldsymbol{x}^{(T)}$ may be farther from the minimizer than the approximation from a similar number of iterations of a more sophisticated, non-differentiable update method. However, the unrolled method's dependence on Ψ is also a benefit, as an unrolled method can be applied to non-smooth cost functions, as long as the resulting update mapping Ψ is smooth. Further, defining Ψ and the initial starting point ensures that $\boldsymbol{x}^{(T)}$ is unique, avoiding concerns about non-unique minimizers.

Another advantage of unrolling is that one can run a given number of iterations of the optimization algorithm, without having to reach convergence, and still calculate a valid gradient. Particularly in image reconstruction problems, where finding $\hat{\boldsymbol{x}}$ exactly can be time intensive, the benefit of a more flexible run-time could outweigh the disadvantages. However, the corresponding downside of unrolling is that the learned hyperparameters are less clearly tied to the original cost function than when one uses the minimizer approach. Section 7.1 further discusses this point in connection to how unrolling for bilevel methods can differ from (deep) learnable optimization algorithms.

One way to connect the minimizer and unrolling strategies is to consider the limit as the number of unrolled iterations approaches infinity. Assuming the optimization algorithm converges, this "fixed point" approach is strongly related to the minimizer approach. For instance, [137] shows that backpropagating through the last $\tilde{T}$ iterations of a converged unrolled algorithm can be viewed as approximating the matrix inverse in the minimizer gradient equation (4.8) with an order-$\tilde{T}$ Taylor series. Section 7.2 further discusses how fixed point networks (or "equilibrium networks") relate the unrolled-to-convergence and minimizer approaches.

Gradient accuracy and computational cost are, unsurprisingly, trade-offs. Tab. 4.1 summarizes the cost of the minimizer and unrolled approaches, derived in Section 4.2.4 and Appendix B respectively, but the total computation will depend on the required gradient accuracy. By

accuracy, we mean error from the true bilevel gradient

$$\| \underbrace{\hat{\nabla}_T \ell(\boldsymbol{\gamma})}_{\substack{\text{Estimated} \\ \text{gradient}}} - \underbrace{\nabla \ell(\boldsymbol{\gamma})}_{\substack{\text{True bilevel} \\ \text{gradient}}} \|,$$

where T denotes the number of lower-level optimization steps. The unrolled gradient is always accurate for the unrolled mapping, but not for the original bilevel problem. Therefore, unrolling may be more computationally feasible when one cannot run a sufficient number of lower-level optimization steps to reach close enough to a minimizer to assume the gradient in (4.3) is approximately zero [29].

In all of the approaches considered, the accuracy of the estimated hyperparameter gradient in turn depends on the solution accuracy or number of unrolled iterations of the lower-level cost function. Ref. [124] notes that their translation to a single level approach failed if they did not optimize the lower-level problem to a sufficient accuracy level.

	Minimizer	Unrolled: reverse	Unrolled: forward
Memory	0	$\mathcal{O}(TN)$	$\mathcal{O}(NR)$
Hessian-vector products	0	$\mathcal{O}(T)$	$\mathcal{O}(TR)$
Hessian-inverse vector products	1	0	0
Other multiplications	NR	$\mathcal{O}(TNR)$	$\mathcal{O}(NR)$

Table 4.1: Memory and computational complexity of the minimizer approach (4.8), reverse-mode unrolled approach (B.2), and forward-mode unrolled approach (B.3) to computing $\nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}; \hat{\boldsymbol{x}}(\boldsymbol{\gamma}))$. Computational costs do not include running the optimization algorithm (typically expensive but often comparable across methods), computing $\nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}; \boldsymbol{x}^{(T)})$ (typically cheap), or computing $\nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}; \boldsymbol{x})$ (frequently zero). Memory requirements do not include storing a single copy of $\boldsymbol{x}$, $\boldsymbol{A}$, $\boldsymbol{\gamma}$, $\boldsymbol{H}$, and $\boldsymbol{J}$. Recall $\boldsymbol{x} \in \mathbb{F}^N$, $\boldsymbol{\gamma} \in \mathbb{F}^R$, and there are T iterations of the lower-level optimization algorithm for the unrolled method. Hessian-vector products (first row) and Hessian-inverse-vector products (middle row) are listed separately from all other multiplications (last row) as the computational cost of Hessian operations can vary widely; see discussion in Section 4.2.4.

However, [123]–[125] did not investigate how the solution accuracy of the lower-level problem impacts the upper-level gradient estimate.

For the minimizer and unrolled approaches, [117], [118] found that the gradient estimate from the minimizer approach converges to the true gradient faster than the unrolled approach (in terms of computation). To state the bounds, [117], [118] assert conditions on the structure of the bilevel problem. They assume that $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ is the unique minimizer of the lower-level cost function, the Hessian of the lower-level is invertible, the Hessian and Jacobian of Φ are Lipschitz continuous with respect to $\boldsymbol{x}$, the gradients of the upper-level loss are Lipschitz continuous with respect to $\boldsymbol{x}$, the norm of $\boldsymbol{x}$ is bounded, and the lower-level cost is strongly convex and Lipschitz smooth for every $\boldsymbol{\gamma}$ value. Section 5.3.1 discusses similar investigations that use these conditions, how easy or hard they are to satisfy, and how they apply to (Ex).

Ref. [118] initializes the lower-level iterates for both the unrolled and minimizer approach with the zero vector, *i.e.*, $\boldsymbol{x}^{(0)} = \mathbf{0}$. Under their assumptions, [118] prove that both the unrolled and minimizer gradients converge linearly in the number of lower-level iterations when the lower-level optimization algorithm and conjugate gradient algorithm for the minimizer approach converge linearly. Although the rate of the approaches is the same, the minimizer approach converges at a faster linear rate and [118] generally recommends the minimizer approach, though they found empirically that the unrolled approach may be more reliable when the strong convexity and Lipschitz smooth assumptions on the lower-level cost do not hold.

Ref. [117] extended the analysis from [118] to consider a warm start initialization for the lower-level optimization algorithm. They similarly find that the minimizer approach has a lower complexity than the unrolled approach. Sections 5.3.2 and 5.3.3 further discuss complexity results after introducing specific bilevel optimization algorithms.

5

Gradient-Based Bilevel Optimization Methods

The previous section discussed different approaches to finding $\nabla \ell(\boldsymbol{\gamma})$, the gradient of the upper-level loss function with respect to the learnable parameters. Building on those results, we now consider approaches for optimizing the bilevel problem. In particular, this section concentrates on gradient-based algorithms for optimizing the hyperparameters. While there is some overlap with single-level optimization methods, this section focuses on the challenges due to the bilevel structure. Therefore, we do not discuss the lower-level optimization algorithms in detail; for overviews of lower-level optimization, see, *e.g.*, [41], [138].

Gradient-based methods for bilevel problems are an alternative to the approaches described in Section 3, *e.g.*, grid or random search, Bayesian optimization, and trust region methods. By incorporating gradient information, the methods presented in this section can scale to problems having many hyperparameters. In fact, Section 5.3 reviews papers that provide bounds on the number of upper-level gradient descent iterations required to reach a point within some user-defined tolerance of a solution. While the bounds depend on the regularity of the upper-level loss and lower-level cost functions, they do not depend on the number of hyperparameters nor the signal dimension. Although

having more hyperparameters will increase computation per iteration, using a gradient descent approach means the number of iterations need not scale with the number of hyperparameters, R.

Bilevel gradient methods fall into two broad categories. Most gradient-based approaches to the bilevel problem fall under the first category: double-loop algorithms. These methods involve (i) optimizing the lower-level cost, either to some convergence tolerance if using a minimizer approach or for a certain number of iterations if using an unrolled approach, (ii) calculating $\nabla \ell(\boldsymbol{\gamma})$, (iii) taking a gradient step in $\boldsymbol{\gamma}$, and (iv) iterating. The first step is itself an optimization algorithm and may involve many inner iterations, thus the categorization as a "double-loop algorithm."

The second category, "single-loop" algorithms, involve one loop, with each iteration containing one gradient step for both the lower-level optimization variable, $\boldsymbol{x}$, and the upper-level optimization variable, $\boldsymbol{\gamma}$. Single-loop algorithms may alternate updates or update the variables simultaneously. Section 4 used t to denote the lower-level iteration counter; this section introduces u as the iteration counter for the upper-level iterations and as the single iteration counter for single-loop algorithms.

5.1 Double-Loop Algorithms

After using one of the approaches in Section 4 to compute the hyperparameter gradient $\nabla \ell(\boldsymbol{\gamma})$, typical double-loop algorithms for bilevel problems run some type of gradient descent on the upper-level loss. Alg. 1 shows an example double-loop algorithm [139]. Line 10 of Alg. 1 uses the CG method to compute the product of the Hessian inverse with a vector in (4.8). Thus, Alg. 1 actually involves three loops. However, the third, CG loop is often left as an implementation detail and we will continue to use the term "double-loop" for the overall strategy. There is similarly a third, hidden loop in approaches that use the reverse mode method for backpropogation in the unrolled approaches described in Section 4.4.

The final iterate of a lower-level optimizer is only an approximation of the lower-level minimizer. However, the minimizer approach to calculating the upper-level gradient $\nabla \ell(\boldsymbol{\gamma})$ from Section 4.2 assumes

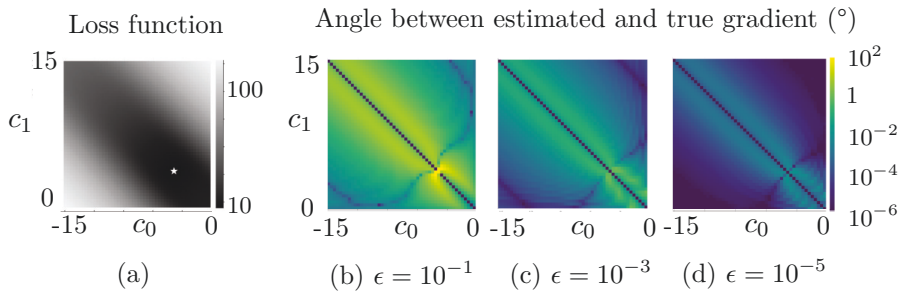


Figure 5.1: Error in the upper-level gradient, $\nabla\ell(\boldsymbol{\gamma})$, for various convergence thresholds for the lower-level optimizer. The bilevel problem is (Ex) with a single filter, $\boldsymbol{c} = \begin{bmatrix} c_0 & c_1 \end{bmatrix}$, $e^{\beta_0} = 0$, $e^{\beta_1} = -5$, and $\phi(z) = z^2$ so there is an analytic solution for $\nabla\ell(\boldsymbol{\gamma})$. The training data is piece-wise constant 1d signals and the learnable hyperparameters are the filter coefficients. (a) Upper-level loss function, $\ell(\boldsymbol{\gamma})$. The cost function is low (dark) where $c_1 \approx c_0$, corresponding to approximate finite differences. The star indicates the minimum. (b-d) Error in the estimated gradient angle using the minimizer approach (4.8), defined as the angle between $\hat{\nabla}\ell(\boldsymbol{\gamma})$ and $\nabla\ell(\boldsymbol{\gamma})$, when the lower-level optimization is run until $\|\nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})\|_2 < \epsilon$.

$\nabla_{\boldsymbol{x}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}) = \mathbf{0}$. Any error stemming from not being at an exact critical point can be magnified in the full calculation (4.8), and the resulting hyperparameter gradient will be an approximation of the true gradient, as illustrated in Fig. 5.1. Thus, how accurately one optimizes the lower-level problem can greatly impact the quality of the learned parameters, $\hat{\boldsymbol{\gamma}}$ [140]. Alternatively, if one uses the unrolled approach with a set number of iterations (4.17), the gradient is accurate for that specific number of iterations, but the lower-level optimization sequence may not have converged and the overall method may not accurately approximate the original bilevel problem.

Due to such inevitable inexactness when computing $\nabla\ell(\boldsymbol{\gamma})$, one may wonder about the convergence of double-loop algorithms for bilevel problems. Considering the unrolled method of computing $\nabla\ell(\boldsymbol{\gamma})$, [141] showed that the sequence of hyperparameter values in a double-loop algorithm, $\boldsymbol{\gamma}^{(u)}$, converges as the number of unrolled iterations increases. To prove this result, [141] assumed the hyperparameters were constrained to a compact set, $\ell(\boldsymbol{\gamma}\,;\boldsymbol{x})$ and $\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ are jointly continuous, there

Algorithm 1 Hyperparameter optimization with approximate gradient (HOAG) from [139]. As written below, the HOAG algorithm is impractical because it uses $\hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(u)})$ in the convergence criteria; however, for strongly convex lower-level problems, the convergence criteria, $\|\hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(u)}) - \boldsymbol{x}^{(t)}(\boldsymbol{\gamma}^{(u)})\|$, is easily upper-bounded.

procedure HOAG($\{\epsilon^{(u)}, u = 1, 2, \ldots\}$, $\boldsymbol{\gamma}^{(0)}$, $\boldsymbol{x}^{(0)}$, $\boldsymbol{y}$)
  for u **do**=0,1,... ▷ Upper-level iteration counter
    $t = 0$ ▷ Lower-level iteration counter
    while $\|\hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(u)}) - \boldsymbol{x}^{(t)}(\boldsymbol{\gamma}^{(u)})\| \geq \epsilon^{(u)}$ **do**
      $\boldsymbol{x}^{(t+1)} = \Psi(\boldsymbol{x}^{(t)}; \boldsymbol{\gamma}^{(u)})$ ▷ Lower-level optimization step
      $t = t + 1$
    end while
    Compute gradient $\nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}^{(u)}; \boldsymbol{x}^{(t)})$ and
    Jacobian $\nabla_{\boldsymbol{x}\boldsymbol{\gamma}} \Phi(\boldsymbol{x}^{(t)}; \boldsymbol{\gamma}^{(u)})$
    Using CG, find $\boldsymbol{q}$ such that
    $\|\nabla_{\boldsymbol{x}\boldsymbol{x}} \Phi(\boldsymbol{x}^{(t)}; \boldsymbol{\gamma}^{(u)})\boldsymbol{q} - \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}^{(u)}; \boldsymbol{x}^{(t)})\| \leq \epsilon^{(u)}$
    $\boldsymbol{g} = \nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}^{(u)}; \boldsymbol{x}^{(t)}) - \left(\nabla_{\boldsymbol{x}\boldsymbol{\gamma}} \Phi(\boldsymbol{x}^{(t)}; \boldsymbol{\gamma}^{(u)})\right)' \boldsymbol{q}$ ▷ From (4.8)
    $\boldsymbol{\gamma}^{(u+1)} = \boldsymbol{\gamma}^{(u)} - \frac{1}{L}\boldsymbol{g}$ ▷ L is a Lipschitz constant of $\nabla \ell(\boldsymbol{\gamma})$
  end for
  return $\boldsymbol{\gamma}^{(u+1)}$
end procedure

is a unique solution $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ to the lower-level cost for all $\boldsymbol{\gamma}$; and $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ is bounded for all $\boldsymbol{\gamma}$. These conditions are satisfied for problems with strictly convex lower-level cost functions and suitable box constraints on $\boldsymbol{\gamma}$. Section 5.3.2 further discusses convergence results for double-loop algorithms.

Pedregosa [139] proved a similar result for the minimizer formula (4.8) using CG to compute (4.10). Specifically, [139] showed that the hyperparameter sequence convergences to a stationary point if the sequence of positive tolerances, $\{\epsilon^{(u)}, u = 1, 2, \ldots\}$ in Alg. 1, is summable. The convergence results are for the algorithm version shown in Alg. 1 that uses a Lipschitz constant of $\ell(\boldsymbol{\gamma})$, which is generally unknown. Although [139] discusses various empirical strategies for setting the step

size, the convergence theory does not consider those variations. Thus, the double-loop algorithm [139] requires multiple design decisions.

There are four key design decisions for double-loop algorithms:

1. How accurately should one solve the lower-level problem?
2. What upper-level gradient descent algorithm should one use?
3. How does one pick the step size for the upper-level descent step?
4. What stopping criteria should one use for the upper-level iterations?

This section first reviews some (largely heuristic) approaches to these design decisions and presents example bilevel gradient descent methods with no (or few) assumptions beyond those made in Section 4. Without any further assumptions, the answers to the questions above are based on heuristics, with few theoretical guarantees but often providing good experimental results. Section 5.3.2 discusses recent methods with stricter assumptions on the bilevel problem and their theory-backed answers to the above questions.

The first step in a double-loop algorithm is to optimize the lower-level cost, for which there are many optimization approaches. The only restriction is computability of the gradient of the upper-level loss $\nabla \ell(\boldsymbol{\gamma})$, which typically includes a smoothness assumption (see Section 4 for discussion). Many bilevel methods use a standard optimizer for the lower-level problem, although others propose new variants, *e.g.*, [34].

The **first design decision** (how accurately to solve the lower-level problem) involves a trade-off between computational complexity and accuracy. Example convergence criteria are fairly standard to the optimization literature, *e.g.*, the Euclidean norm of the lower-level gradient [30], [142] or the normalized change in the estimate $\boldsymbol{x}$ [143] being less than some threshold. For example, [30] used a convergence criteria of $\|\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}^{(t)}; \boldsymbol{\gamma})\|_2 \leq 10^{-3}$ (where the image scale is 0-255). As mentioned above, [139] uses a sequence of convergence tolerances so that the lower-level cost function is optimized more accurately as the upper-level iterations continue.

Ref. [140] investigated the importance of lower-level optimization accuracy. The authors use the same training model as in [31], which is the bilevel extension of the Field of Experts [55], but varied the convergence criteria for the lower-level problem. When using a convergence tolerance

of $\|\nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x}^{(t)};\boldsymbol{\gamma})\|_2/\sqrt{N} \leq 10^{-5}$, [140] found an average improvement of 0.65dB in the PSNR for test images over [31], who ran their lower-level optimization algorithm for a set number of iterations. Ref. [140] also plots the test PSNR and training loss versus the lower-level convergence criteria and shows how test PSNR increases and training loss decreases with increased lower-level solution accuracy for this specific filter learning bilevel problem.

Many publications do not report a specific threshold or discuss how they chose a convergence criteria or number of lower-level iterations. However, a few note the importance of such decisions. For example, [124] found that their learning method fails if the lower-level optimizer is insufficiently close to the minimizer and [30] stated their results are "significantly better" than [31] because they solve the lower-level problem "with high[er] accuracy."

After selecting a level of accuracy, finding (an approximation of) $\hat{\boldsymbol{x}}$, and calculating $\nabla\ell(\boldsymbol{\gamma})$ using one of the approaches from Section 4, one must make the **second design decision**: which gradient-based method to use for the upper-level problem. Many bilevel methods suggest a simple gradient-based method such as plain gradient descent (GD) [33], GD with a line search (see the third design decision), projected GD [132], or stochastic GD [124]. These methods update $\boldsymbol{\gamma}$ based on only the current upper-level gradient; they do not have memory of previous gradients nor require/estimate any second-order information.

Methods that incorporate some second-order information use more memory and computation per iteration, but may converge faster than basic GD methods. For example, Broyden-Fletcher-Goldfarb-Shanno (BFGS) and L-BFGS (the low-memory version of BFGS) [144] are quasi-Newton algorithms that store and update an approximate Hessian matrix that serves as a preconditioner for the gradient. The $R \times R$ size of the Hessian grows as the number hyperparameters increases, but quasi-Newton methods like L-BFGS use practical rank-1 updates with storage $\mathcal{O}(R)$. Adam [145] is a popular GD method, especially in the machine learning community, that tailors the step size (equivalently the learning rate) for each hyperparameter based on moments of the gradient. Although Adam requires its own parameters, the parameters are relatively easy to set and the default settings often perform adequately.

Example bilevel papers using methods with second-order information include those that use BFGS [32], L-BFGS [30], Gauss-Newton [146], and Adam [34].

Many gradient-based methods require selecting a step size parameter, *e.g.*, one must choose a step size α_ℓ in classical GD:

$$\boldsymbol{\gamma}^{(u+1)} = \boldsymbol{\gamma}^{(u)} - \alpha_\ell \nabla \ell \left(\boldsymbol{\gamma}^{(u)} \right).$$

This choice is the **third design decision**. Bilevel problems are generally non-convex, and typically a Lipschitz constant is unavailable, so line search strategies initially appear appealing. However, any line search strategy that involves attempting multiple values quickly becomes computationally intractable for large-scale problems. The upper-level loss function in bilevel problems is particularly expensive to evaluate because it requires optimizing the lower-level cost! Further, recall that the upper-level loss is typically an expectation over multiple training samples (UL), so evaluating a single step size involves optimizing the lower-level cost J times (or using a stochastic approach and selecting a batch size).

Despite these challenges, a line search strategy may be viable if it rarely requires multiple attempts. For example, the backtracking line search in [142] that used the Armijo–Goldstein condition required 57-59 lower-level evaluations (per training example) over 40 upper-level gradient descent steps, so most upper-level steps required only one lower-level evaluation. Other bilevel papers that used backtracking with Armijo-type conditions include [11], [32], [143]; [147] used the Barzilai-Borwein method for picking an adaptive step size.

Other approaches to determining the step size are: (i) normalize the gradient by the dimension of the data and pick a fixed step size [124], (ii) pick a value that is small enough based on experience [33], or (iii) adapt the step size based on the decrease from the previous iteration [139].

The **fourth design decision** is the convergence criteria for the upper-level loss. As with the lower-level convergence criteria, few publications include a specific threshold, but most bilevel methods tend to use traditional convergence criteria such as the norm of the hyperparameter gradient falling below some threshold [32], the norm of the

change in parameters falling below some threshold [30], and/or reaching a maximum iteration count (many papers). One specific example is to terminate when the normalized change in learned parameters, $\|\boldsymbol{\gamma}^{(u+1)} - \boldsymbol{\gamma}^{(u)}\| / \|\boldsymbol{\gamma}^{(u)}\|$, is below 0.01 [143]. The normalized change bound is convenient because it is unitless and thus invariant to scaling of $\boldsymbol{\gamma}$.

Fig. 5.2 shows example upper-level convergence plots for a double-loop algorithm for the bilevel problem (Ex). After an initial first run of OGM to get the lower-level initialization $\hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(0)})$ such that $\frac{1}{\sqrt{N}} \left\| \nabla_{\boldsymbol{x}} \Phi \left(\hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(0)}) ; \boldsymbol{\gamma}^{(0)} \right) \right\|_2 < 10^{-7}$, the lower-level optimizer consisted of 10 iterations of OGM [148], initialized with the estimate from the previous upper-level iteration. The upper-level optimizer is Adam [145] with the default parameters, negating the need for a separate upper-level step-size parameter. We ran 10,000 outer-loop iterations. The final norm of the upper-level gradient, $\frac{1}{\sqrt{R}} \|\nabla \ell(\boldsymbol{\gamma})^{(U)}\|$ was 0.08 when learning the filter coefficients and tuning parameters and $5 \cdot 10^{-4}$ when learning only $\boldsymbol{\beta}$. Fig. 6.2 shows the corresponding denoised images and Appendix D.2 further details the experiment settings.

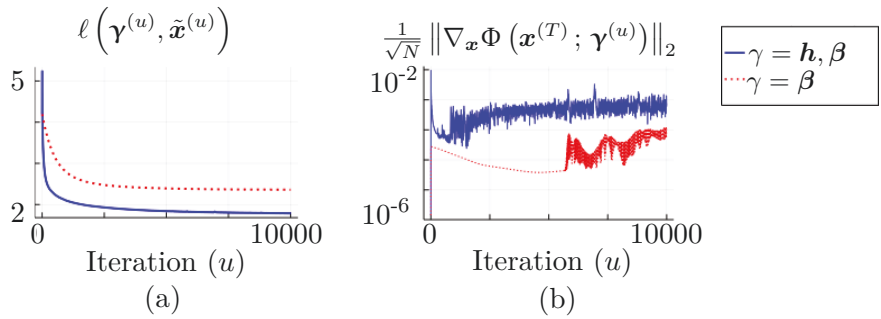


Figure 5.2: Example convergence plots for a double-loop bilevel method when $\boldsymbol{\gamma}$ includes $\boldsymbol{h}$ and $\boldsymbol{\beta}$ (solid lines) and when $\boldsymbol{\gamma} = \boldsymbol{\beta}$ (dotted lines). (a) Estimated upper-level loss function evaluated at the current estimate of the lower-level minimizer, $\boldsymbol{x}^{(T)} = \boldsymbol{x}^{(T)}(\boldsymbol{\gamma}^{(u)})$, versus upper-level iteration u. (b) Lower-level convergence metric, averaged over all training samples, versus upper-level iteration. The estimated lower-level minimizer remains close to convergence throughout the double-loop method.

5.2 Single-Loop Algorithms

Unlike double-loop algorithms, single-loop algorithms take a gradient step in $\boldsymbol{\gamma}$ without optimizing the lower-level problem each step. Two early bilevel method papers [26], [60] proposed single-loop approaches based on solving the system of equations that arises from the Lagrangian.

The system of equations approach in [26], [60] closely follows the KKT perspective on the minimizer approach in Section 4.2.2. Recall that the gradient of the lower-level problem is zero at a minimizer, $\hat{\boldsymbol{x}}$, and one can use this equality as a constraint on the upper-level loss function. The corresponding Lagrangian is

$$L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) = \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}) + \boldsymbol{\nu}' \nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}), \tag{5.1}$$

where $\boldsymbol{\nu}$ is a vector of Lagrange multipliers. For the filter learning example (Ex), the Lagrangian is

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}^{\text{true}}\|_2^2 + \\ \boldsymbol{\nu}' \left(\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) + e^{\beta_0} \sum_{k=1}^{K} e^{\beta_k} \tilde{\boldsymbol{c}}_k \circledast \phi.(\boldsymbol{c}_k \circledast \boldsymbol{x}; \epsilon) \right). \end{aligned}$$

As in Section 4.2.2, we consider derivatives of the Lagrangian with respect to $\boldsymbol{\nu}$, $\boldsymbol{x}$, and $\boldsymbol{\gamma}$. Here are the general expressions and the specific equations for the filter learning example (Ex) when considering the element of $\boldsymbol{\gamma}$ corresponding to β_k:

$$\begin{aligned} \nabla_{\boldsymbol{\nu}} L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) &= \nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}) \\ &= \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) + e^{\beta_0} \sum_{k=1}^{K} e^{\beta_k} \tilde{\boldsymbol{c}}_k \circledast \phi.(\boldsymbol{c}_k \circledast \boldsymbol{x}; \epsilon) \\ \nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) &= \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}) + \nabla_{\boldsymbol{x}\boldsymbol{x}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}) \boldsymbol{\nu} \\ &= \boldsymbol{x} - \boldsymbol{x}^{\text{true}} \\ \nabla_{\boldsymbol{\gamma}} L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) &= \nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}) + \boldsymbol{\nu}' \nabla_{\boldsymbol{x}\boldsymbol{\gamma}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}) \\ &= \boldsymbol{\nu}' \left(e^{\beta_0} e^{\beta_k} \tilde{\boldsymbol{c}}_k \circledast \dot{\phi}.(\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}}) \right) \text{ when } \boldsymbol{\gamma} = \beta_k. \end{aligned}$$

These expressions are equivalent to the primal, adjoint, and optimality conditions respectively in [60].

Here the minimizer and single-loop approach diverge. Section 4.2.2 used the above Lagrangian gradients to solve for $\hat{\boldsymbol{\nu}}$, substitute $\hat{\boldsymbol{\nu}}$ into the gradient of the Lagrangian with respect to $\boldsymbol{\gamma}$, and thus find the minimizer expression for $\nabla \ell(\boldsymbol{\gamma})$. The single-loop approach instead considers solving the system of gradient equations directly:

$$\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) = \begin{bmatrix} \nabla_{\boldsymbol{\nu}} L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) \\ \nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) \\ \nabla_{\boldsymbol{\gamma}} L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\nu}) \end{bmatrix} = \mathbf{0}.$$

For example, [60] proposed a Newton algorithm using the Jacobian of the gradient matrix $\boldsymbol{G}$.

Another approach to single-loop algorithms is to replace the "while" loop in Alg. 1 line 4 with a single gradient step in the lower-level optimization variables. Two single-loop algorithms are the two-timescale stochastic approximation (TTSA) method [149] and the Single Timescale stochAstic BiLevEl optimization (STABLE) method [150]. Alg. 2 shows TTSA as an example single-loop algorithm. Both TTSA and STABLE alternate between one gradient step for the lower-level cost and one gradient step for the upper-level problem.

There are two main challenges in designing such a single loop algorithm for bilevel optimization. Because both TTSA and STABLE use the minimizer approach (4.8) to finding the upper-level gradient, the first challenge is ensuring the current lower-level iterate is close enough to the minimizer to calculate a useful upper-level gradient. TTSA addresses this challenge by taking larger steps for the lower-level problem while STABLE addresses this using a lower-level update that better predicts the next lower-level minimizer, $\hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(u+1)})$.

The second main challenge is estimating the upper-level gradient, even given stochastic estimates of $\nabla_{\boldsymbol{xx}} \Phi$ and $\nabla_{\boldsymbol{x\gamma}} \Phi$, because the minimizer equation (4.8) is nonlinear. The theoretical results about TTSA are built on the assumption that the upper-level gradient is biased due to this nonlinearity. In contrast, STABLE uses recursion to update estimates of the gradients and thus reduce variance. Section 5.3.3 goes into more detail about both algorithms.

Algorithm 2 Two-Timescale Stochastic Approximation (TTSA) method from [149]. TTSA includes a possible projection of the hyperparameter after each gradient step onto a constraint set, not shown here. The tildes denote stochastic approximations for the corresponding expressions.

procedure TTSA($\boldsymbol{\gamma}^{(0)}$, $\boldsymbol{x}^{(0)}$, $\alpha_\ell^{(u)}$, $\alpha_\Phi^{(u)}$)
  for $u = 1, \ldots$ **do**
    $\boldsymbol{x}^{(u+1)} = \boldsymbol{x}^{(u)} - \alpha_\Phi^{(u)} \tilde{\nabla}_{\boldsymbol{x}} \Phi(\boldsymbol{x}^{(u)}; \boldsymbol{\gamma}^{(u)})$
    $\boldsymbol{g} = \nabla_{\boldsymbol{\gamma}} \ell^{(u)} - \left(\tilde{\nabla}_{\boldsymbol{x}\boldsymbol{\gamma}} \Phi^{(u)}\right)' \left(\tilde{\nabla}_{\boldsymbol{x}\boldsymbol{x}} \Phi^{(u)}\right)^{-1} \nabla_{\boldsymbol{x}} \ell^{(u)}$
    $\boldsymbol{\gamma}^{(u+1)} = \boldsymbol{\gamma}^{(u)} - \alpha_\ell^{(u)} \boldsymbol{g}$
  end for
end procedure

5.3 Complexity Analysis

A series of recent papers established finite-time sample complexity bounds for stochastic bilevel optimization methods based on gradient descent for the upper-level loss and lower-level cost. Ref.s [117], [151] use double-loop approaches and [149], [150] use single-loop algorithms. Unlike most of the methods discussed in Section 5.1, these papers make additional assumptions about the upper and lower-level functions then select the upper and lower-level step sizes to ensure convergence.

In these works, "finite-time sample complexity" refers to big-O bounds on a number of iterations that ensures one reaches a minimizer to within some desired tolerance. In contrast to asymptotic convergence analysis, finite-time bounds provide information about the estimated hyperparameters, $\boldsymbol{\gamma}^{(u)}$, after a finite number of upper-level iterations. These bounds depend on problem-specific quantities, such as Lipschitz constants, but not on the hyperparameter or signal dimensions.

To summarize the results, this section returns to the notation from the introduction where the upper-level loss may be deterministic or

		Upper-level gradients	Lower-level gradients
Double-loop	BA	$\mathcal{O}\left(\log\left(\frac{1}{\epsilon^2}\right)\right)$	$\mathcal{O}\left(\log\left(\frac{1}{\epsilon^3}\right)\right)$
	stocBiO	$\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$
Single-loop	TTSA	$\mathcal{O}\left(\frac{1}{\epsilon^{2.5}}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon^{2.5}}\right)$
	STABLE	$\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$

Table 5.1: Finite-time sample complexities for the stochastic bilevel problem in the common scenario where ℓ is non-convex when using BA [151], stocBiO [117], TTSA [149], and STABLE [150]. When ℓ is strongly convex, the sample complexity of STABLE is $\mathcal{O}\left(\frac{1}{\epsilon^1}\right)$ (for the upper- and lower-level gradients), which is the same as single level stochastic gradient algorithms. See cited papers for other complexity results when ℓ is strongly convex.

stochastic, *e.g.*, the bilevel problem is

$$\hat{\boldsymbol{\gamma}} = \operatorname*{argmin}_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}) \text{ with } \ell(\boldsymbol{\gamma}) = \begin{cases} \ell(\boldsymbol{\gamma}, \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) & \text{deterministic} \\ \mathbb{E}\left[\ell(\boldsymbol{\gamma}, \hat{\boldsymbol{x}}(\boldsymbol{\gamma}))\right] & \text{stochastic.} \end{cases} \tag{5.2}$$

The expectation in (5.2) can have different meanings depending on the setting. When one has J training images with one noise realization per image, one often picks a random subset ("minibatch") of those J images for each update of $\boldsymbol{\gamma}$, corresponding to stochastic gradient descent of the upper-level loss. In this setting, the randomness is a property of the algorithm, not of the upper-level loss, and the expectation reduces to the deterministic case. Section 7.5 discusses other possible definitions of the stochastic bilevel formulation.

The complexity results (summarized in Tab. 5.1) are all in terms of finding $\boldsymbol{\gamma}_\epsilon$, defined as an ϵ-optimal solution. In the (atypical) setting where $\ell(\boldsymbol{\gamma})$ is convex, $\boldsymbol{\gamma}_\epsilon$ is an ϵ-optimal solution if it satisfies either $\ell(\boldsymbol{\gamma}_\epsilon) - \ell(\hat{\boldsymbol{\gamma}}) \leq \epsilon$ [117], [149], [151] or $\|\hat{\boldsymbol{\gamma}} - \boldsymbol{\gamma}_\epsilon\|^2 \leq \epsilon$ [150]. (These conditions are equivalent if ℓ is strongly convex in $\boldsymbol{\gamma}$, but can differ otherwise.) In the (common) non-convex setting, $\boldsymbol{\gamma}_\epsilon$ is typically called an ϵ-stationary point if it satisfies $\|\nabla \ell(\boldsymbol{\gamma}_\epsilon)\|^2 \leq \epsilon$ [117], [150], [151]. In the stochastic setting, $\boldsymbol{\gamma}_\epsilon$ must satisfy these conditions in expectation.

The following sections briefly describe the BA, stocBiO, TTSA, and STABLE algorithms. The literature in this area is quickly evolving;

between the writing and editing of this work, new double-loop and single-loop methods appeared with improved complexity results. For example, [152], [153] concurrently proposed bilevel optimization methods that leverage momentum and variance reduction techniques to reduce the bound on the number of iterations to $\widetilde{\mathcal{O}}\left(\frac{1}{\epsilon^{1.5}}\right)$ for both upper-level and lower-level gradients. Ref. [152] achieved this complexity result for both a double-loop method and a single-loop method.

Whether double-loop or single-loop methods are preferred is an open question. Refs. [117], [152] find that double-loop methods converge faster (in terms of wall time) than single-loop methods. The authors hypothesize that $\nabla\ell(\boldsymbol{\gamma})$ is sensitive enough to changes in the estimate of the lower-level optimizer that the increased accuracy of the double-loop estimates of $\nabla\ell(\boldsymbol{\gamma})$ is worth the additional lower-level optimization time. Future work should test this hypothesis in different experimental settings and establish guidelines on when to use a double-loop or single-loop algorithm.

5.3.1 Assumptions

References [117], [149]–[151] all make similar assumptions about ℓ and Φ to derive theoretical results for their proposed bilevel optimization methods. We first summarize the set of sufficient conditions from [151], and later note any additional assumptions used by the other methods. The conditions in [151] on the upper-level function, $\ell(\boldsymbol{\gamma}\,;\,\boldsymbol{x})$, are:

- Aℓ1. $\forall\boldsymbol{\gamma}\in\mathbb{F}^R$, $\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma},\boldsymbol{x})$ and $\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma},\boldsymbol{x})$ are Lipschitz continuous with respect to $\boldsymbol{x}$, with corresponding Lipschitz constants $L_{\boldsymbol{x},\nabla_{\boldsymbol{\gamma}}\ell}$ and $L_{\boldsymbol{x},\nabla_{\boldsymbol{x}}\ell}$. (These constants are independent of $\boldsymbol{x}$ and $\boldsymbol{\gamma}$.)
- Aℓ2. The gradient with respect to $\boldsymbol{x}$ is bounded, *i.e.*, $\|\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma},\boldsymbol{x})\|\leq C_{\nabla_{\boldsymbol{x}}\ell},\ \forall\boldsymbol{x}\in\mathbb{F}^N$.
- Aℓ3. $\forall\boldsymbol{x}\in\mathbb{F}^N$, $\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma},\boldsymbol{x})$ is Lipschitz continuous with respect to $\boldsymbol{\gamma}$, with corresponding Lipschitz constant $L_{\boldsymbol{\gamma},\nabla_{\boldsymbol{x}}\ell}$.

The conditions in [151] on the lower-level function, $\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$, are:

- AΦ1. Φ is continuously twice differentiable in $\boldsymbol{\gamma}$ and $\boldsymbol{x}$.

AΦ2. $\forall \boldsymbol{\gamma} \in \mathbb{F}^R$, $\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ is Lipschitz continuous with respect to $\boldsymbol{x}$ with corresponding constant $L_{\boldsymbol{x},\nabla_{\boldsymbol{x}}\Phi}$.

AΦ3. $\forall \boldsymbol{\gamma} \in \mathbb{F}^R$, $\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ is strongly convex with respect to $\boldsymbol{x}$, *i.e.*, $\mu_{\boldsymbol{x},\Phi} \boldsymbol{I} \preceq \nabla^2_{\boldsymbol{x}} \Phi(\boldsymbol{\gamma}\,;\boldsymbol{x})$, for some $\mu_{\boldsymbol{x},\Phi} > 0$.

AΦ4. $\forall \boldsymbol{\gamma} \in \mathbb{F}^R$, $\nabla_{\boldsymbol{xx}} \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ and $\nabla_{\boldsymbol{\gamma x}} \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ are Lipschitz continuous with respect to $\boldsymbol{x}$ with Lipschitz constants $L_{\boldsymbol{x},\nabla_{\boldsymbol{xx}}\Phi}$ and $L_{\boldsymbol{x},\nabla_{\boldsymbol{\gamma x}}\Phi}$.

AΦ5. The mixed second gradient of Φ is bounded, *i.e.*, $\|\nabla_{\boldsymbol{\gamma x}} \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})\| \leq C_{\nabla_{\boldsymbol{\gamma x}}\Phi}, \quad \forall \boldsymbol{\gamma}, \boldsymbol{x}$.

AΦ6. $\forall \boldsymbol{x} \in \mathbb{F}^N$, $\nabla_{\boldsymbol{\gamma x}} \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ and $\nabla_{\boldsymbol{xx}} \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ are Lipschitz continuous with respect to $\boldsymbol{\gamma}$ with Lipschitz constants $L_{\boldsymbol{\gamma},\nabla_{\boldsymbol{\gamma x}}\Phi}$ and $L_{\boldsymbol{\gamma},\nabla_{\boldsymbol{xx}}\Phi}$.

In addition to the assumptions above on ℓ and Φ, analyses of optimization algorithms for the stochastic bilevel problem assume that (i) all estimated gradients are unbiased and (ii) the variance of the estimation errors is bounded by $\sigma^2_{\nabla_{\boldsymbol{\gamma}}\ell}$, $\sigma^2_{\nabla_{\boldsymbol{x}}\ell}$, $\sigma^2_{\nabla_{\boldsymbol{x}}\Phi}$, $\sigma^2_{\nabla_{\boldsymbol{\gamma x}}\Phi}$, and $\sigma^2_{\nabla_{\boldsymbol{xx}}\Phi}$. The stochastic methods discussed here are all based on the minimizer approach to finding the upper-level gradient. Therefore, the methods use estimates of $\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\,\boldsymbol{x})$, $\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\,\boldsymbol{x})$, $\nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$, $\nabla_{\boldsymbol{\gamma},\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$, and $\nabla_{\boldsymbol{x},\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$. We denote the estimates of these gradient using tildes, *e.g.*, $\tilde{\nabla}_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\,\boldsymbol{x})$. Following (4.8), an estimate of the upper-level gradient approximation is thus

$$\hat{\nabla}\ell(\boldsymbol{\gamma}) = \tilde{\nabla}_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma},\boldsymbol{x}) - (\tilde{\nabla}_{\boldsymbol{x\gamma}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}))'(\tilde{\nabla}_{\boldsymbol{xx}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}))^{-1}\tilde{\nabla}_{\boldsymbol{x}}\ell(\boldsymbol{\gamma},\boldsymbol{x}).$$

As an example of the bounded variance assumption, [151] assumes

$$\mathbb{E}\left[\|\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\,\boldsymbol{x}) - \hat{\nabla}_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x})\|^2\right] \leq \sigma^2_{\nabla_{\boldsymbol{\gamma}}\ell} \quad \forall \boldsymbol{x}, \boldsymbol{\gamma}.$$

To consider how the complexity analysis bounds may apply in practice, Appendix C.2 examines how assumptions Aℓ1-Aℓ3 and assumptions AΦ1-AΦ6 apply to the running filter learning example (Ex). Although a few of the conditions are easily satisfied, most are not. Appendix C.2 shows that the conditions are met if one invokes box constraints on the variables $\boldsymbol{x}$ and $\boldsymbol{\gamma}$. Although imposing box constraints requires modifying the algorithms, *e.g.*, by including a projection step, the iterates

remain unchanged if the constraints are sufficiently generous. However, such generous box constraints are likely to yield large Lipschitz constants and bounds, leading to overly-conservative predicted convergence rates. Further, any differentiable upper-level loss and lower-level cost function would met the conditions above with such box constraints. Generalizing the following complexity analysis for looser conditions is an important avenue for future work.

5.3.2 Double-loop

Ghadimi and Wang [151] were the first to provide a finite-time analysis of the bilevel problem. The authors proposed and analyzed the Bilevel Approximation (BA) method (see Alg. 3). BA uses two nested loops. The inner loop minimizes the lower-level cost to some accuracy, determined by the number of lower-level iterations; the more inner iterations, the more accurate the gradient will be, but at the cost of more computation and time. The outer loop is (inexact) projected gradient steps on ℓ. Ref. [151] used the minimizer result (4.8) (with the IFT perspective for the derivation) to estimate the upper-level gradient.

To bound the complexity of BA, [151] first related the error in the lower-level solution to the error in the upper-level gradient estimate as

$$\|\underbrace{\hat{\nabla}_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}, \boldsymbol{x}^{(T)})}_{\text{Estimated gradient}} - \underbrace{\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}, \hat{\boldsymbol{x}}(\boldsymbol{\gamma}))}_{\text{True gradient}}\| \leq C_{\text{GW}} \underbrace{\left\|\boldsymbol{x}^{(T)} - \hat{\boldsymbol{x}}(\boldsymbol{\gamma})\right\|}_{\text{Error in lower-level}},$$

where C_{GW} is a constant that depends on many of the bounds defined in the assumptions above [151]. Combing the above error bound with known gradient descent bounds for the accuracy of the lower-level problem yields bounds on the accuracy of the upper-level gradient. The standard lower-level bounds can vary by the specific algorithm ([151] uses plain GD), but are in terms of $Q_\Phi = \frac{L_{\boldsymbol{x},\nabla_{\boldsymbol{x}}\Phi}}{\mu_{\boldsymbol{x},\Phi}}$ (the "condition number" for the strongly convex lower-level function) and the distance between the initialization and the minimizer.

Ref. [151] shows that $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ is Lipschitz continuous in $\boldsymbol{\gamma}$ under the above assumptions, which intuitively states that the lower-level minimizer does not change too rapidly with changes in the hyperparameters. Further, $\nabla\ell(\boldsymbol{\gamma})$ is Lipschitz continuous in $\boldsymbol{\gamma}$ with a Lipschitz constant,

Algorithm 3 Bilevel Approximation (BA) Method from [151]. The differences for the AID-BiO and ITD-BiO methods from [117] are: (1) when $u > 0$, the BiO methods replace line 3 with $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(T_{u-1})}$, (2) T_i does not vary with upper-level iteration, (3) the upper-level gradient calculation in line 7 can use the minimizer approach (4.8) or backpropagation (B.2), and (4) the hyperparameter update is standard gradient descent, so line 8 becomes $\boldsymbol{\gamma}^{(u+1)} = \boldsymbol{\gamma}^{(u)} - \alpha_\ell \boldsymbol{g}$.

1: **procedure** BA($\boldsymbol{\gamma}^{(0)}$, $\boldsymbol{x}^{(0)}$, α_ℓ, α_Φ, T_u $\forall u$)
2: **for** $u = 1, \ldots$ **do** ▷ Upper-level iterations
3: $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(0)}$ ▷ Included for comparison with [117]
4: **for** $t = 1 : T_u$ **do** ▷ T lower-level iterations
5: $\boldsymbol{x}^{(t)} = \boldsymbol{x}^{(t-1)} - \alpha_\Phi \nabla_{\boldsymbol{x}} \Phi(\boldsymbol{\gamma}, \boldsymbol{x}^{(t-1)})$
6: **end for**
7: $\boldsymbol{g} = \nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}^{(u)}, \boldsymbol{x}^{T_i})$ ▷ Use minimizer result (4.8)
8: $\boldsymbol{\gamma}^{(u+1)} = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\boldsymbol{\gamma} - \boldsymbol{\gamma}^{(u)}\|^2 + \alpha_\ell \langle \boldsymbol{g}, \boldsymbol{\gamma} \rangle \right\}$
9: **end for**
10: **end procedure**

$L_{\boldsymbol{\gamma}, \nabla_{\boldsymbol{\gamma}} \ell}$, that depends on many of the constants given above.

The main theorems from [151] hold when the lower-level GD step size is $\alpha_\Phi = \frac{2}{L_{\boldsymbol{x}, \nabla_{\boldsymbol{x}} \Phi} + \mu_{\boldsymbol{x}, \Phi}}$ and the upper-level step size satisfies $\alpha_\ell \leq \frac{1}{L_{\boldsymbol{\gamma}, \nabla_{\boldsymbol{\gamma}} \ell}}$. Then, the distance between the uth loss function value and the minimum loss function value, $\ell(\boldsymbol{\gamma}^{(u)}, \hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(u)}) - \ell(\hat{\boldsymbol{\gamma}}, \hat{\boldsymbol{x}}(\hat{\boldsymbol{\gamma}}))$, is bounded by a constant that depends on the starting distance from a minimizer (dependent on the initialization of $\boldsymbol{\gamma}$ and $\boldsymbol{x}$), Q_Φ, C_{GW}, the number of inner iterations, and the upper-level step size. The bound differs for strongly convex, convex, and possibly non-convex upper-level loss functions. Tab. 5.2 summarizes the sample complexity required to reach an ϵ-optimal point in each of these scenarios.

Ji, Yang, and Liang [117] proposed two methods for Bilevel Optimization that improve on the sample complexities from [151] for non-convex loss functions under similar assumptions. The first, ITD-BiO (ITerative Differentiation), uses the unrolled method for calculating the upper-level gradient (see Section 4.4). The second, AID-BiO (Approximate Implicit

$\ell(\boldsymbol{\gamma})$	Upper-level gradients	Lower-level gradients
Strongly convex	$\mathcal{O}\left(\log\left(\frac{1}{\epsilon}\right)\right)$	$\mathcal{O}\left(\log^2(\frac{1}{\epsilon}\right)$
Convex	$\mathcal{O}\left(\frac{1}{\sqrt{\epsilon}}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon^{3/4}}\right)$
Non-convex	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon^{5/4}}\right)$

Table 5.2: Sample complexity to reach an ϵ-optimal solution of the deterministic bilevel problem using BA [151], for various assumptions on the upper-level loss function. Usually $\ell(\boldsymbol{\gamma})$ is non-convex and that case has the worst-case order results. The complexities show the total number of partial gradients of the upper-level loss (equal to the number of lower-level Hessians needed for estimating $\nabla\ell(\boldsymbol{\gamma})$ using (4.8)) and the partial gradients of the lower-level. The convex results use the accelerated BA method, which uses acceleration techniques similar to Nesterov's method [154] applied to the upper-level gradient step in Alg. 3.

Differentiation), uses the minimizer method with the implicit function theory perspective (see Section 4.2). Tab. 5.3 summarizes the sample complexities [117]. Much of the computational advantage of ITD-BiO and AID-BiO is in improving the iteration complexity with respect to the condition number (not shown in the summary table).

One of the main computational advantages of the AID-BiO and IFT-BiO methods in [117] over the BA algorithm Alg. 3 is a warm restart for the lower-level optimization. Although the hyperparameters change every outer iteration, the change is generally small enough that the stopping point of the previous lower-level descent is a better initialization than the noisy data (recall that [151] showed the lower-level minimizer is Lipschitz continuous in $\boldsymbol{\gamma}$). One can account for this warm restart when using automatic differentiation tools (backpropagation) [117]. The caption for Alg. 3 summarizes the other differences between BA and the BiO methods.

The Bilevel Stochastic Approximation (BSA) method replaces the lower-level update in BA (see Alg. 3) with standard stochastic gradient descent. The corresponding upper-level step in BSA is a projected gradient step with stochastic estimates of all gradients. Another difference in the stochastic versions of the BA [151] and BiO [117] methods is that they use an inverse matrix theorem (based on the Neumann series) to

	Upper-level gradients	Lower-level gradients	Hessian-vector products
BA	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon^{5/4}}\right)$	$\widetilde{\mathcal{O}}\left(\frac{1}{\epsilon}\right)$
AID-BiO	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$
ITD-BiO	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$	$\widetilde{\mathcal{O}}\left(\frac{1}{\epsilon}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$

Table 5.3: A comparison of the finite-time sample complexity to reach an ϵ-solution of the deterministic bilevel problem when the upper-level loss function is non-convex using BA [151], AID-BiO [117], and ITD-BiO [117]. $\widetilde{\mathcal{O}}(\cdot)$ = order omits any $\log(\epsilon)^{-1}$ term.

estimate the Hessian inverse. Ref. [117] simplifies the inverse Hessian calculation to replace expensive matrix-matrix multiplications with matrix-vector multiplications. This same strategy makes backpropagation more computationally efficient than the forward mode computation for the unrolled gradient; see Appendix B.

5.3.3 Single-Loop

Recently, [149], [150] extended the double-loop analysis of [117], [151] to single-loop algorithms that alternate gradient steps in $\boldsymbol{x}$ and $\boldsymbol{\gamma}$.

Alg. 2 summarizes the single-loop algorithm TTSA [149]. The analysis of TTSA uses the same lower-level cost function assumptions as mentioned above for BSA [151] and one additional upper-level assumption: that ℓ is weakly convex with parameter μ_ℓ, *i.e.*,

$$\ell(\boldsymbol{\gamma}+\boldsymbol{\delta}) \geq \ell(\boldsymbol{\gamma})\langle\nabla\ell(\boldsymbol{\gamma}),\boldsymbol{\delta}\rangle + \mu_\ell\left\|\boldsymbol{\delta}\right\|^2, \quad \forall\boldsymbol{\gamma},\boldsymbol{\delta}\in\mathbb{R}^R.$$

TTSA assumes the lower-level gradient estimate is still unbiased and that its variance is now bounded as

$$\mathbb{E}\left[\left\|\nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x},\boldsymbol{\gamma}) - \tilde{\nabla}_{\boldsymbol{x}}\Phi(\boldsymbol{x},\boldsymbol{\gamma})\right\|^2\right] \leq \sigma^2_{\nabla_{\boldsymbol{x}}\Phi}\left(1+\left\|\nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x},\boldsymbol{\gamma})\right\|^2\right).$$

Further, the stochastic upper-level gradient estimate, $\tilde{\nabla}_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}^{(u)},\boldsymbol{x}^{(u+1)})$, includes a bias that stems from the nonlinear dependence on the lower-level Hessian. This bias decreases as the batch size increases.

The "two-timescale" part of TTSA comes from using different upper and lower step size sequences. The lower-level step size is larger and

bounds the tracking error (the distance between $\hat{\boldsymbol{x}}$ and the $\boldsymbol{x}$ iterate) as the hyperparameters change (at the upper-level loss's relatively slower rate). Thus, [149] chose step-sizes such that $\alpha_\ell(u)/\alpha_\Phi(u) \to 0$. Specifically, if ℓ is strongly convex, then α_ℓ is $\mathcal{O}(u^{-1})$ and α_Φ is $\mathcal{O}\left(u^{-2/3}\right)$. If ℓ is convex, then α_ℓ is $\mathcal{O}\left(u^{-3/4}\right)$ and α_Φ is $\mathcal{O}\left(u^{-1/2}\right)$.

Chen, Sun, and Yin [150] improved the sample complexity of TTSA. By using a single timescale, their algorithm, STABLE, achieves the "same order of sample complexity as the stochastic gradient descent method for the single-level stochastic optimization" [150]. However, the improved sample complexity comes at the cost of additional computation per iteration as STABLE can no longer trade a matrix inversion (of size $R \times R$) for matrix-vector products, as done in the [117]. Ref. [150] therefore recommended STABLE when sampling is more costly than computation or when R is relatively small.

The analysis of STABLE uses the same upper-level loss and lower-level cost function assumptions as listed above for BSA. Additionally, STABLE assumes that, $\forall \boldsymbol{x}$, $\nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,;\boldsymbol{x})$ is Lipschitz continuous in $\boldsymbol{\gamma}$. This condition is easily satisfied as many upper-level loss functions do not regularize $\boldsymbol{\gamma}$. Further, those that do often use a squared 2-norm, *i.e.*, Tikhonov-style regularization, that has a Lipschitz continuous gradient. Additionally, rather than bounding the gradient norms as in assumptions Aℓ 2 and AΦ5, [149] assumes the following moments are bounded:

- the second and fourth moment of $\nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,;\boldsymbol{x})$ and $\nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,;\boldsymbol{x})$ and
- the second moment of $\nabla_{\boldsymbol{\gamma}\boldsymbol{x}} \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$ and $\nabla_{\boldsymbol{x}\boldsymbol{x}} \Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$,

ensuring that the upper-level gradient is Lipschitz continuous.

Like the previous algorithms discussed, STABLE evaluates the minimizer result (4.8) at non-minimizer lower-level iterates, $\boldsymbol{x}^{(T)}(\boldsymbol{\gamma}^{(u)})$, to estimate the hyperparameter gradient. However, it differs in how it estimates and uses the gradients. STABLE replaces the upper-level gradient in TTSA line 4 with

$$\boldsymbol{g} = \nabla_{\boldsymbol{\gamma}} \ell^{(u)} - \underbrace{(\Delta_{\boldsymbol{x}\boldsymbol{\gamma}}^{(u)})'}_{\text{Prev. } \tilde{\nabla}_{\boldsymbol{x}\boldsymbol{\gamma}} \Phi^{(u)}} \underbrace{(\Delta_{\boldsymbol{x}\boldsymbol{x}}^{(u)})^{-1}}_{\text{Prev. } \tilde{\nabla}_{\boldsymbol{x}\boldsymbol{x}} \Phi^{(u)}} \nabla_{\boldsymbol{x}} \ell^{(u)}. \tag{5.3}$$

Taking inspiration from variance reduction techniques for single-level optimization problems, *e.g.*, [155], STABLE recursively updates the

newly defined matrices as follows:

$$\Delta_{\boldsymbol{x}\boldsymbol{\gamma}}^{(u)} = \mathcal{P}_{\|\Delta\| \leq C_{\nabla_{\boldsymbol{x}\boldsymbol{\gamma}}\Phi}} \left((1-\tau_u) \underbrace{(\Delta_{\boldsymbol{x}\boldsymbol{\gamma}}^{(u-1)} - \tilde{\nabla}_{\boldsymbol{x}\boldsymbol{\gamma}}\Phi^{(u-1)})}_{\text{Recursive update}} + \underbrace{\tilde{\nabla}_{\boldsymbol{x}\boldsymbol{\gamma}}\Phi^{(u)}}_{\text{New estimate}} \right)$$

$$\Delta_{\boldsymbol{x}\boldsymbol{x}}^{(u)} = \mathcal{P}_{\Delta \succeq \mu_{\boldsymbol{x},\Phi} \boldsymbol{I}} \left((1-\tau_u) \overbrace{(\Delta_{\boldsymbol{x}\boldsymbol{x}}^{(u-1)} - \tilde{\nabla}_{\boldsymbol{x}\boldsymbol{x}}\Phi^{(u-1)})} + \overbrace{\tilde{\nabla}_{\boldsymbol{x}\boldsymbol{x}}\Phi^{(u)}} \right).$$

In the $\Delta_{\boldsymbol{x}\boldsymbol{\gamma}}^{(u)}$ update, the projection onto the set of matrices with a maximum norm helps ensure stability by not allowing the gradient to get too large. The projection in the $\Delta_{\boldsymbol{x}\boldsymbol{x}}^{(u)}$ update is an eigenvalue truncation that ensures positive definiteness of the estimated Hessian in this Newton-based method. After computing the gradient $\boldsymbol{g}$ (5.3), the upper-level update is a standard descent step as in Alg. 2 line 5.

STABLE [150] also uses the recursively estimated gradient matrices in the lower-level cost function descent. It replaces the standard gradient descent step in Alg. 2 line 3 with one that uses second order information:

$$\boldsymbol{x}^{(u+1)} = \boldsymbol{x}^{(u)} - \underbrace{\alpha_\Phi(u)\tilde{\nabla}_{\boldsymbol{x}}\Phi(\boldsymbol{x}^{(u)}; \boldsymbol{\gamma}^{(u)})}_{\text{Standard GD step}} - \underbrace{(\Delta_{\boldsymbol{x}\boldsymbol{x}}^{(u)})^{-1}(\Delta_{\boldsymbol{\gamma}\boldsymbol{x}}^{(u)})'(\boldsymbol{x}^{(u+1)} - \boldsymbol{x}^{(u)})}_{\text{New term}}.$$

With these changes, STABLE is able to reduce the iteration complexity relative to TTSA as summarized in Tab. 5.1.

5.4 Summary of Methods

There are many variations of gradient-based methods for optimizing bilevel problems, especially when one considers that many of the upper-level descent strategies can work with either the minimizer or unrolled approach discussed in Section 4. There is no clear single "best" algorithm for all applications; each algorithm involves trade-offs.

Building on the minimizer and unrolled methods for finding the upper-level gradient with respect to the hyperparameters, $\nabla\ell(\boldsymbol{\gamma})$, double-loop algorithms are an intuitive approach. Although optimizing the lower-level problem every time one takes a gradient step in $\boldsymbol{\gamma}$ is computationally expensive, the lower-level problem is is embarrassingly parallelizable across samples. Specifically, one can optimize the lower-level cost for each training sample independently before averaging the

resulting gradients to take an upper-level gradient step. In the typical scenario when training is performed offline, training wall-time can therefore be dramatically reduced by using multiple processors.

Single-loop algorithms remove the need to optimize the lower-level cost function multiple times. The single-loop algorithms that consider a system of equations often accelerate convergence using Newton solvers [11], [60]. However, the optimality system grows quickly when there are multiple training images, and may become too computationally expensive as J increases [30]. Another type of single-loop algorithm uses alternating gradient steps in $\boldsymbol{x}$ and $\boldsymbol{\gamma}$ [149], [150]. Although each method has slight variations (such as whether it uses momemtum), these single-loop methods are generally equivalent to considering $T = 1$ in the double-loop methods.

This section organized algorithms based on the number of for-loops; double-loop algorithms have two loops while single-loop algorithms have one[1]. However, there are many other ways in which bilevel optimization methods differ and not all methods fall cleanly into one group. One such example is the Penalty method [156]. The Penalty method forms a single-level, constrained optimization problem, with the constraint that the gradient of the lower-level cost function should be zero, $\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}) = \mathbf{0}$. (This step is similar to the derivation of the minimizer approach via KKT conditions; see Section 4.2.2.) Rather than forming the Lagrangian as in (5.1), [156] penalizes the norm of the gradient, with increasing penalties as the upper iterations increase. Thus, the Penalty cost function[2] at iteration u is

$$p(\boldsymbol{\gamma}\,, \boldsymbol{x}) = \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) + \lambda^{(u)} \left\| \nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma}) \right\|_2^2 .$$

The penalty variable sequence, $\lambda^{(u)}$, must be positive, non-decreasing, and divergent ($\lambda^{(u)} \to \infty$).

Penalty [156] incorporates elements of both double-loop and single-loop algorithms. Similar to the double-loop algorithms, Penalty takes multiple gradient descent steps in the lower-level optimization variable, $\boldsymbol{x}$, before calculating and updating the hyperparameters. However,

[1] As noted at the start of the section, this loop counting does not include the loop in CG or in backpropagation

[2] This is a simplification; [156] allows for constraints on $\boldsymbol{x}$ and $\boldsymbol{\gamma}$.

Penalty forms a single-level optimization problem that could be optimized using techniques such as those used in single-loop algorithms.

Another variant on a double-loop bilevel optimization method is to optimize a lower-level surrogate function $\tilde{\Phi}(\boldsymbol{x}\,;\boldsymbol{\gamma}^{(u)})$ instead of optimizing $\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}^{(u)})$. For example, [157] replaces Φ with its first-order approximation around the current solution point $(\boldsymbol{\gamma}^{(u)}, \hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(u)}))$. Because this approximation is only reliable in the neighborhood of $(\boldsymbol{\gamma}^{(u)}, \hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(u)}))$, [157] adds the proximal term $\lambda\|\boldsymbol{\gamma}-\boldsymbol{\gamma}^{(u)}\|^2$ to the upper-level loss function at each outer iteration, where λ is a positive tuning parameter.

The finite-time complexity analyses [117], [149]–[152] justify the use of gradient-based bilevel methods for problems with many hyperparameters, as none of the sample complexity bounds involved the number of hyperparameters. This is in stark contrast with the hyperparameter optimization strategies in Section 3. However, the per-iteration cost for bilevel methods is still large and increasing with the hyperparameter dimension. Further, the conditions on the lower-level cost function AΦ 1-AΦ6 seem restrictive and may not be satisfied in practice. Complexity analysis based on more relaxed conditions could be very valuable.

Because of the restrictive conditions in the complexity analysis, it is generally infeasible to compute theoretically justified step-sizes and other algorithm parameters in the single-loop and double-loop methods [117], [149]–[152]. Thus, one must often resort to grid searches or use heuristics, such as those discussed in Section 5.1, to select these algorithm parameters. Ref. [152] comments on one example of how empirical practice can differ from theory. Although their theory requires that the number of iterates of the Neumann series used to approximate the inverse Hessian matrix grows with the desired solution accuracy, the authors found that using a few iterates was sufficient (and faster) in practice.

Gradient-based and other hyperparameter optimization methods are active research areas, and the trade-offs continue to evolve. Although it currently seems that gradient-based bilevel methods make sense for problems with many hyperparameters, new methods may overtake or combine with what is presented here. For example, many bilevel methods (and convergence analyses thereof) use classical gradient descent for the lower-level optimization algorithm, whereas [158] showed that the

Optimized Gradient Method (OGM) has better convergence guarantees and is optimal among first-order methods for smooth convex problems [159]. These advances provide opportunities for further acceleration of bilevel methods.

6

Survey of Applications

Bilevel methods have been used in many image reconstruction applications, including 1D signal denoising [33], image denoising (see following sections), compressed sensing [34], spectral CT image reconstruction [143], and MRI image reconstruction [34]. Bilevel methods are also used for classification problems. For example, [160, Sec. 6] shows how the structured support vector machine (SSVM) is a convex surrogate for the bilevel model when the lower-level cost is linear in $\boldsymbol{\gamma}$. This section discusses trends and highlights specific applications to provide concrete examples of bilevel methods for image reconstruction.

Many papers present or analyze bilevel optimization methods for general upper-level loss functions and lower-level cost functions, under some set of assumptions about each level. Sections 4 and 5 summarized many of these methods. Although there are cases when the choice of a loss function and/or cost impacts the optimization strategy, many bilevel problems could use any optimization method. Thus, this section concentrates on the specific applications, rather than methodology.

This section is split into a discussion of lower-level cost and upper-level loss functions. (Lower-level cost functions that involve CNNs are discussed separately; see Section 7.1.) The conclusion section discusses

examples where the loss function is tightly connected to the cost function.

6.1 Lower-level Cost Function Design

Once a bilevel problem is optimized to find $\hat{\boldsymbol{\gamma}}$, the learned parameters are typically deployed in the same lower-level problem as used during training but with new, testing data. Thus, it is the lower-level cost function that specifies the application of the bilevel problem, *e.g.*, CT image reconstruction or image deblurring.

Denoising applications consider the case where the forward model is an identity operator ($\boldsymbol{A} = \boldsymbol{I}$). This case has the simplest possible data-fit term in the cost function and requires the least amount of computation when computing gradients or evaluating Φ. Because bilevel methods are generally already computationally expensive, it is unsurprising that many papers focus on denoising, even if only as a starting point towards applying the proposed bilevel method to other applications.

More general image reconstruction problems consider non-identity forward models. Few papers learn parameters for image reconstruction in the fully task-based manner described in (UL), likely due to the additional computational cost. Some papers, *e.g.*, [29], [30], [68] consider learning parameters for denoising, and then apply $\hat{\boldsymbol{\gamma}}$ in a reconstruction problem with the same regularizer but introducing the new $\boldsymbol{A}$ to the data-fit term. These "crossover experiments" [68] test the generalizability of the learned parameters, but they sacrifice the specific task-based nature of the bilevel method.

Recall from Section 2 that the regularizer (with its learned parameters) can be related to a prior for $\boldsymbol{x}$ in a maximum *a posteriori* probability perspective. If this perspective is valid, then the $\hat{\boldsymbol{\gamma}}$ should generalize to other system matrices. However, the exact connection between the regularizer and the probability distribution is not straightforward [161] and previous results suggest that $\hat{\boldsymbol{\gamma}}$ varies with different $\boldsymbol{A}$'s [25], [68]. Further, $\boldsymbol{A}$ often is an imperfect model for the true underlying phenomena and $\hat{\boldsymbol{\gamma}}$ may end up compensating for modeling errors that are specific to a given $\boldsymbol{A}$, and thus may not generalize to other imaging system models.

Many bilevel methods, especially in image denoising [30], [31], [33], [60], [146], but also in image reconstruction [32], use the same or a very similar lower-level cost as the running example in this review. From Section 1.2, the running example cost function is:

$$\hat{\boldsymbol{x}}(\boldsymbol{\gamma}, \boldsymbol{y}) = \underset{\boldsymbol{x}}{\operatorname{argmin}} \overbrace{\frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + e^{\beta_0} \underbrace{\sum_{k=1}^{K} e^{\beta_k} \mathbf{1}' \phi(\boldsymbol{c}_k \circledast \boldsymbol{x}; \epsilon)}_{R(\boldsymbol{x};\boldsymbol{\gamma})}}^{\Phi(\boldsymbol{x};\boldsymbol{\gamma})} . \tag{6.1}$$

The learned hyperparameters, $\boldsymbol{\gamma}$, include the tuning parameters, β_k and/or the filter coefficients, $\boldsymbol{c}_k$. The image reconstruction example in [32] generalized (6.1) for implicitly defined forward models by using a different data-fit term, as given in (4.7). Their two example problems involve learning parameters to estimate the diffusion coefficient or forcing function in a second-order elliptic partial differential equation.

Two common variations among applications using (6.1) are (1) the choice of which tuning parameters to learn and (2) what sparsifying function, ϕ, to use. Some methods [32], [60], [146] learn only the tuning parameters; these methods typically use finite differencing filters or discrete cosine transform (DCT) filters (excluding the DC filter) as the $\boldsymbol{c}_k$'s. Other methods learn only filter coefficients [33]. Fig. 6.1 shows filters learned from patches of the "cameraman" image when $\boldsymbol{\gamma} = (\boldsymbol{\beta}, \boldsymbol{h})$ and shows filter strengths when $\boldsymbol{\gamma} = \boldsymbol{\beta}$. The corresponding bilevel problem is (Ex) with ϕ given in (CR1N). Fig. 6.2 shows the corresponding denoised image and Appendix D.2 describes the experiment settings and additional results.

A slight variation on learning the filters is to learn coefficients for a linear combination of filter basis elements [30], [31], *i.e.*, learning $a_{k,i}$ where

$$\boldsymbol{c}_k = \sum_i a_{k,i} \boldsymbol{b}_i,$$

for some set of basis filter elements, $\boldsymbol{b}_i$. One benefit of imposing a filter basis is the ability to ensure the filters lie in a given subspace. For example, [30], [31] use the DCT as a basis and remove the the constant filter so that all learned filters are guaranteed to have zero-mean.

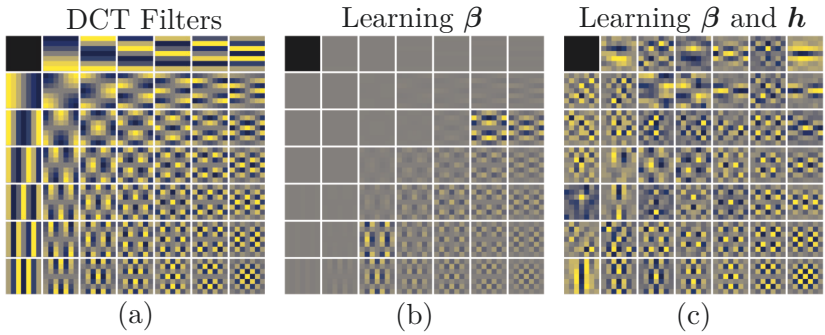


Figure 6.1: The DCT filter bank and example learned filters for (Ex) with training data from the "cameraman" image. (a) The 48 non-constant 7×7 DCT filters used to initialize $\boldsymbol{\gamma}$. The dark, top-left square represents the removed DC filter. (b) The DCT filters multiplied by their respective tuning parameter β_k when $\boldsymbol{\gamma} = \boldsymbol{\beta}$. The range of $e^{\beta_0+\beta_k}$ is 0.001-1.08. The learned tuning parameters emphasize the higher-frequency DCT filters. (c) Learned filters when $\boldsymbol{\gamma} = (\boldsymbol{\beta}, \boldsymbol{h})$ (scaled to have unit-norm for visualization).

In terms of sparsifying functions, [33], [146] used the same corner rounded 1-norm as in (CR1N), [31] used $\phi = \log\left(1 + z^2\right)$ to relate their method to the Field of Experts framework [55], [32] used a quadratic penalty, and [30], [60] both consider multiple ϕ options to examine the impact of non-convexity in ϕ. Ref. [60] compared p-norms, $\|\boldsymbol{c}_k \circledast \boldsymbol{x}\|_p^p$, for $p \in \{\frac{1}{2}, 1, 2\}$, where the $p = \frac{1}{2}$ and $p = 1$ cases are corner-rounded to ensure ϕ is smooth. (The $p = \frac{1}{2}$ case is non-convex.) Ref. [30] compared the convex corner-rounded 1-norm in (CR1N) with two non-convex choices: the log-sum penalty $\log\left(1 + z^2\right)$, and the Student-t function $\log\left(10\epsilon + \sqrt{z^2 + \epsilon^2}\right)$.

Both [30], [60] found that non-convex penalty functions led to denoised images with better (higher) PSNR. They hypothesize that the improvement is due to the non-convex penalty functions better matching the heavy-tailed distributions in natural images. As further evidence of the importance of non-convexity, [30] found that untrained 7×7 DCT filters (excluding the constant filter) with learned tuning parameters and a non-convex ϕ outperformed learned filter coefficients with a convex ϕ, despite the increased data adaptability when learning

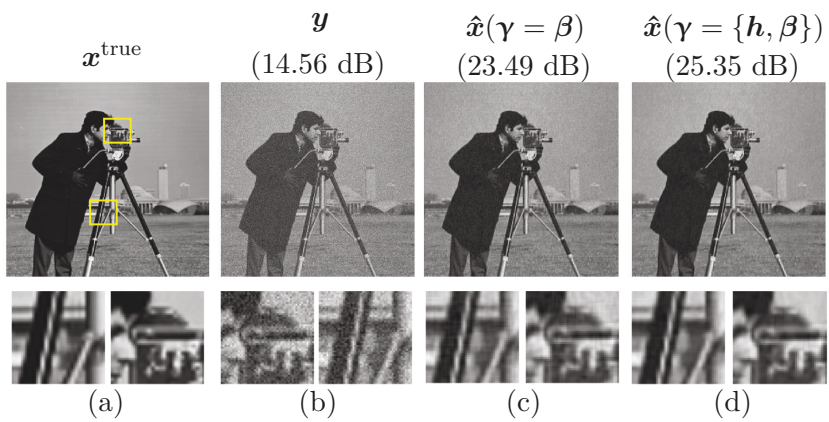


Figure 6.2: Example denoising results for the full "cameraman" test image and two of the training patches. (a) Noiseless training "cameraman" test image. (b) Noisy image and its SNR. (c) Denoised image using the learned tuning parameters that weight the DCT filters as shown in Fig. 6.1b. (d) Denoised image using the learned filter coefficients and tuning parameters as shown in Fig. 6.1c. For comparison, the denoised image using BM3D [162] has a SNR of 26.87. See Appendix D.2 for more details.

filter coefficients. The trade-off for using non-convex penalty functions is the possibility of local minimizers of the lower-level cost.

Chen, Ranftl, and Pock [30] also investigated how the number of learned filters and the size of the filters impacted denoising PSNR. They concluded that increasing the number of filters to achieve an over-complete filter set may not be worth the increased computational expense and that increasing the filter size past 11×11 is unlikely to improve PSNR. Using 48 filters of size 7×7 and the log-sum penalty function, [30] achieved denoising results on natural images comparable to algorithms such as BM3D [162], as seen in Fig. 6.3. Although results will vary between applications and training data sets, the results from [30] provide motivation for filter learning and an initial guide for designing bilevel methods.

In addition to variations on the running example for Φ (6.1), a common regularizer for the lower-level cost is Total Generalized Variation with order 2 (TGV^2) [163]. Whereas TV encourages images to be piece-wise constant, TGV^2 is a generalization of TV designed for

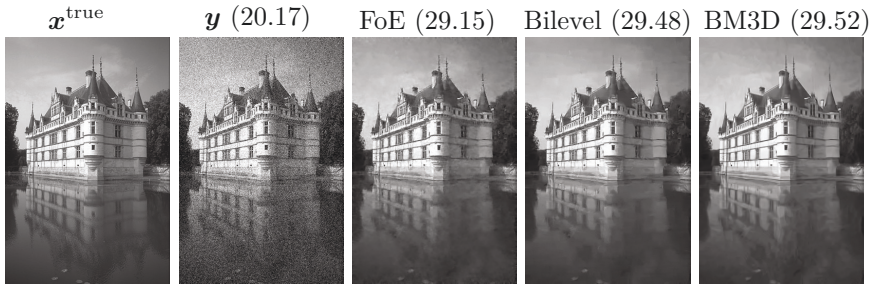


Figure 6.3: Example denoising results from [30] comparing filters learned using bilevel methods to other denoising methods. (a) The original image $\boldsymbol{x}^{\text{true}}$. (b) The noisy image $\boldsymbol{y}$. (c-d) Denoised images using FoE [55], BM3D [162], and a bilevel approach using a set-up equivalent to (Ex) with a non-convex penalty function, $\phi(z) = \log\left(1 + z^2\right)$ [30]. The PSNR values in dB are given in parenthesis. ©2014 IEEE. Reprinted, with permission, from [30].

piece-wise linear images. Another generalization of TV for piece-wise linear images is Infimal Convolutional Total Variation (ICTV) [164]. Bilevel papers that investigate ICTV include [11], [12]; these papers also investigate TGV^2. See [165] for a comparison of the two.

TGV cost functions are typically expressed in the continuous domain, at least initially, but then discretized for implementation, *e.g.*, [166], [167]. One discrete approximation of the TGV^2 regularizer is:

$$R_{\text{TGV}}(\boldsymbol{x}) = \min_{\boldsymbol{z}} e^{\beta_1} \left\|\boldsymbol{c}_{\text{TV}} \circledast \boldsymbol{x} - \boldsymbol{z}\right\|_1 + e^{\beta_2} \left\|\partial \boldsymbol{z}\right\|_1,$$

where $\boldsymbol{c}_{\text{TV}}$ is a filter that takes finite differences and ∂ is a filter that approximates a symmetrized gradient. In TV, one usually thinks of $\boldsymbol{z}$ as a sparse vector; here $\boldsymbol{z}$ is a vector whose finite differences are sparse, so $\boldsymbol{z}$ is approximately piece-wise constant. Encouraging $\boldsymbol{z}$ to be piece-wise constant in turn makes $\boldsymbol{x}$ approximately piece-wise linear, since $\boldsymbol{c}_{\text{TV}} \circledast \boldsymbol{x} \approx \boldsymbol{z}$ from the first term. Bilevel methods for learning β_1 and β_2 for the TGV^2 regularizer include [11], [12]. An extension to the TGV^2 regularizer model is to learn a space-varying tuning parameter [142].

As an example of how the regularizer should be chosen based on the application, [142] found that standard TV with a learned tuning parameter performed best (in terms of SSIM) for approximately piece-

wise constant images while TGV^2 with learned tuning parameters performed best for approximately piece-wise linear images.

6.2 Upper-Level Loss Function Design

From some of the earliest bilevel methods, *e.g.*, [26], [33], to some of the most recent bilevel methods, *e.g.*, [29], [132], square error or mean squared error (MSE) remains the most common upper-level loss function. In the unsupervised setting, [93], [94] used SURE (an estimate of the MSE, see Section 3.1) as the upper-level loss function. Unlike many perceptually motivated image quality measures, MSE is convex in $\boldsymbol{x}$ and it is easy to find $\nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}; \hat{\boldsymbol{x}}(\boldsymbol{\gamma}))$. However, MSE does not capture perceptual quality nor image utility (see Section 3.1). This section discusses a few bilevel methods that used different loss functions.

Ref. [12] compared a squared error upper-level loss function with a Huber (corner rounded 1-norm) loss function. The corresponding lower-level problem was a denoising problem with a standard 2-norm data-fit term and three different options for a regularizer: TV, TGV^2, and ICTV. The authors learned tuning parameters for a natural image dataset using both upper-level loss function options for each of the lower-level regularizers.

Since SNR is equivalent to MSE, the MSE loss will always perform the best according to any SNR-based metric (assuming the bilevel model is well-trained). However, [12] found the tuning parameters learned using the Huber loss yielded denoised images with better qualitative properties and better SSIM, especially at low noise levels. Like MSE, the Huber loss operates point-wise and is easy to differentiate. Thus, the authors conclude that the Huber loss is a good trade-off between tractability and improving on MSE as an image quality measure.

A set of loss functions in [142], [143], [146] consider the unsupervised or "blind" bilevel setting, where one wishes to reconstruct an image without clean samples. Therefore, rather than using an image quality metric that compares a reconstructed image, $\hat{\boldsymbol{x}}$, to some true image, $\boldsymbol{x}^{\text{true}}$, these loss function consider the estimated residual,

$$\hat{\boldsymbol{n}} = \hat{\boldsymbol{n}}(\boldsymbol{\gamma}) = \boldsymbol{A}\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) - \boldsymbol{y},$$

where $\boldsymbol{\gamma}$ is learned using only noisy data. Unsupervised bilevel methods may be beneficial when there is no clean data and one has more knowledge of noise properties than of expected image content. All three methods [142], [143], [146] assume the noise variance, σ^2, is known.

The earliest example [146], learned tuning parameters $\boldsymbol{\gamma}$ such that $\hat{\boldsymbol{n}}$ matched the second moment of the assumed Gaussian distribution for the noise. Their lower-level cost is comparable to (Ex), but re-written in terms of $\boldsymbol{n}$ and with pre-defined finite differencing or 5×5 DCT filters, *i.e.*, they learn only the tuning parameters, β_k. Their upper-level loss encourages the empirical variances of the noise in different frequency bands to match the expected variances:

$$\ell(\boldsymbol{\gamma}\,; \boldsymbol{n}(\boldsymbol{\gamma})) = \frac{1}{2} \sum_i \frac{\left(\|\boldsymbol{f}_i \circledast \boldsymbol{n}\|_2^2 - \mu_i \right)^2}{v_i}$$

$$\mu_i = \mathbb{E}\left[\|\boldsymbol{f}_i \circledast \boldsymbol{n}\|_2^2 \right] \text{ and } v_i = \mathrm{Var}\left[\|\boldsymbol{f}_i \circledast \boldsymbol{n}\|_2^2 \right],$$

where $\boldsymbol{f}_i$ are predetermined filters that select specific frequency components. By using bandpass filters that partition Fourier space, the corresponding means and variances of the second moments of the filtered noise are easily computed, with

$$\mu_i = N\sigma^2 \|\boldsymbol{f}_i\|^2 \quad \text{and} \quad v_i = N\sigma^4 \|\boldsymbol{f}_i\|^4 .$$

Although the experimental results are promising, [146] does not claim state-of-the-art results since their lower-level denoiser is relatively simple.

As an alternative to the Gaussian-inspired approach in [146], [142] and [143] use loss functions that penalize noise outside a set "noise corridor." Both methods learn space-varying tuning parameters, and the upper-level loss consists of a data-fit term (that measures noise properties) and a regularizer on $\boldsymbol{\gamma}$. The data-fit term in the upper-level loss function in [146] defines the noise corridor between a maximum variance, $\bar{\sigma}^2$, and a minimum variance, $\underline{\sigma}^2$:

$$\mathbf{1}' F.\left(\boldsymbol{w} \odot (\boldsymbol{n}(\boldsymbol{\gamma}) \odot \boldsymbol{n}(\boldsymbol{\gamma}))\right) \text{ for}$$
$$F(n) = \frac{1}{2}\max(n - \bar{\sigma}^2, 0)^2 + \frac{1}{2}\min(n - \underline{\sigma}^2, 0)^2, \tag{6.2}$$

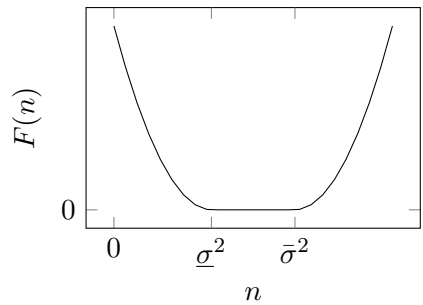


Figure 6.4: Noise corridor function (6.2) used as part of the upper-level loss function for the unsupervised bilevel method in [142].

where $\boldsymbol{w}$ is a predetermined weighting vector. The noise corridor function, $F(n)$, penalizes any noise outside of the expected range as shown in Fig. 6.4. Ref. [143] uses the same noise corridor function, but extends the bilevel method for images with Poisson noise; [143] thus estimates the noisy image using the Kullback-Leibler distance. In addition to the noise corridor function as the data-fit component of the upper-level loss function, [142], [143] include a smoothness-promoting regularizer on $\boldsymbol{\gamma}$, which is a spatially varying tuning parameter vector in both methods.

The task-based nature of bilevel typically makes regularizers or constraints on $\boldsymbol{\gamma}$ unnecessary (see Section 7.4 for common options for other forms of learning). However, there are two general cases where a regularizer on $\boldsymbol{\gamma}$ is useful in the upper-level loss function. First, a regularizer can help avoid over-fitting when the amount of training data is insufficient for the number of learnable hyperparameters. This is often the case when learning space-varying parameters that have similar dimensions as the input data, *e.g.*, [26], [142], [143], [168]. In such cases, the regularization often takes the form of a 2-norm on the learned hyperparameters, $\|\boldsymbol{\gamma}\|_2^2$.

Second, some problems require application-specific constraints, *e.g.*, [68] incorporates constraints in the upper-level loss to ensure that the learned parameters are valid interpolation kernels. Many other hyperparameter constraints do not require a regularization term, For example, non-negativity constraints on tuning parameters are easily handled by redefining the tuning parameter in terms of an exponential, as in (Ex),

and box constraints are common and easy to incorporate with a projection step if using a gradient-based method. Constraints that require sparsity on the learned parameters may benefit from regularization in the upper-level loss function.

An example of an application-specific constraint is found in [27], [28], which consider MRI reconstruction with a data-fit term and a variational regularizer. Both papers extend the bilevel model in (Ex) to include part of the forward model in the learnable parameters, $\boldsymbol{\gamma}$. Specifically, [27], [28] learned the sparse sampling matrix for MRI. (Ref. [28] additionally learns tuning parameters for predetermined filters, whereas [27] sets the tuning parameters and filters and learns only the sampling matrix.) Here, the forward model is

$$\boldsymbol{A} = \operatorname{diag}\Big(\underbrace{s_1, s_2, \ldots, s_M}_{\boldsymbol{s}(\boldsymbol{\gamma})} \Big) \boldsymbol{F},$$

where $\boldsymbol{F}$ is the DFT matrix and s_i are learned binary values that specify whether a frequency location should be sampled.

The motivation for learning a sparse sampling matrix comes from the lower-level MRI reconstruction problem; designing more effective sparse sampling patterns in MRI can decrease scan time and thus improve patient experience, decrease cost, and decrease artifacts from patient movement. This goal requires the learned parameters, s_i, to be binary, which in turn influences the upper-level loss function design. Thus, [27], [28] include regularization in the upper-level to encourage $\boldsymbol{s}$ to be sparse, *e.g.*, [28] uses an upper-level loss with a squared error term and regularizer on $\boldsymbol{s}$:

$$\ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})) = \|\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) - \boldsymbol{x}^{\text{true}}\|_2^2 + \lambda \sum_i \left(s_i + s_i(1 - s_i)\right), \tag{6.3}$$

where λ is a upper-level tuning parameter that one must set manually. (In experiments, they thresholded the learned s_i values to be exactly binary.) An alternative approach is to constrain the number of samples [169], though that formulation requires other optimization methods.

6.3 Conclusion

This section split the discussion of lower-level cost and upper-level loss functions to discuss trends in both areas. However, when designing a bilevel problem, design decisions can impact both levels. For example, the unsupervised nature of [143], [146] clearly impacted their choice of upper-level loss function to use noise statistics rather than squared error calculated with ground-truth data. Since it can be challenging to learn many good parameters from noisy training data, the unsupervised nature also likely impacted the authors' decision to learn only tuning parameters and set the filters manually. Another example of coupling between lower-level and upper-level design is when one enforces application-specific constraints on the learned parameters, *e.g.*, using a regularizer like (6.3) in the upper-level loss to promote sparsity of the MRI sampling matrix [27], [28].

In addition to design decisions influencing both levels, bilevel methods may adopt common techniques for the upper-level loss function and lower-level cost function. For example, a common theme is the tendency to use smooth functions, such as replacing the 1-norm with a corner-rounded 1-norm. This approach requires setting a smoothing parameter, *e.g.*, ϵ in (CR1N), which in turn impacts the Lipschitz constant and optimization speed. More accurate approximations generally lead to larger Lipschitz constants and slower convergence. One approach to trading-off the accuracy of the smoothing with optimization speed is to use a graduated approach and approximate the non-smooth term more and more closely as the optimization progresses [34].

The prevalence of smoothing is unsurprising considering that this review focuses on gradient-based bilevel methods. Rare exceptions include [124], [125], which used the (not corner-rounded) one-norm to define ϕ to learn convolutional filters using the translation to a single level approach described in Section 4.3. The impact of smoothing and how accurately one should approximate a non-differentiable point remains an open question.

From an image quality perspective, ideally one would independently design the lower-level cost function and upper-level training loss. The lower-level cost would depend on the imaging physics and would in-

corporate regularizers that expected to provide excellent image quality when tuned appropriately, and the upper-level loss would use terms that are meaningful for the imaging tasks of interest. As we have seen, in practice one often makes compromises to facilitate optimization and reduce computation time.

7

Connections and Future Directions

This final section connects bilevel methods with related approaches and mentions some additional future directions beyond those already described in previous sections.

Shlezinger *et al.* [170] recently proposed a framework, summarized in Fig. 7.1, for categorizing learning-based approaches that combine inferences, or prior knowledge[1], and deep learning. Inferences can include information about the structure of the forward model, $\boldsymbol{A}$, or about the object $\boldsymbol{x}$ being imaged. For example, any known statistical properties of the object of interest could be used to design a regularizer that encourages the minimizer $\hat{\boldsymbol{x}}$ to be compatible with that prior information. At one extreme, inference-based approaches rely on a relatively small number of handcrafted regularizers with a few, if any, tuning parameters learned from training data. At the other extreme, fully learned approaches assume no information about the application or data and learn all hyperparameters from training data.

Ref. [170] proposes two general categories for methods that mix elements of inference-based and learning-based methods. The first cate-

[1]Ref. [170] uses the term "model-based", but this review uses "inferences" to differentiate from other definitions of model-based learning in the literature.

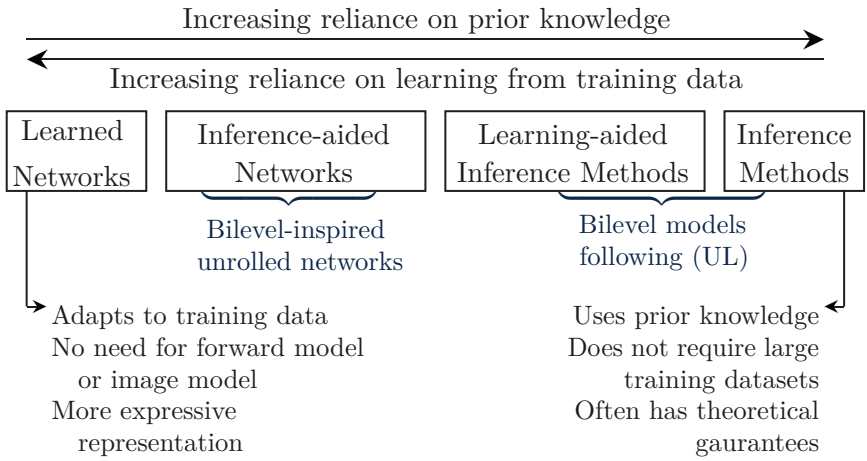


Figure 7.1: Spectrum of learning to inference-based methods from [170].

gory, inference-aided networks, includes deep neural networks (DNNs) with architectures based on an inference-based method. For example, in deep unrolling, one starts with a fixed number of iterations of an optimization algorithm derived from a cost function and then learns parameters that may vary between iterations, or "layers," or may be shared across such iterations. Section 7.1 further discusses unrolling, which is a common inference-aided network design strategy, and the connection to the bilevel unrolling method described in Section 4.4.

The second general category is DNN-aided inference methods [170]. These methods incorporate a deep learning component into traditional inference-based techniques (typically a cost function in image reconstruction). The learned DNN component(s) can be trained separately for each iteration or end-to-end. Because prior knowledge takes a larger role than in the inference-aided networks, these methods typically require smaller training datasets, with the amount of training data required varying with the number of hyperparameters. Section 7.3 discusses how bilevel methods compare to Plug-and-Play, which is an example DNN-aided inference model.

While [170] focused on DNNs due to their highly expressive nature and the abundance of interest in them, the idea of trading off prior

knowledge and learning components applies to machine learning more broadly. Sections 7.1 and 7.3 describe how bilevel methods fit into the framework from [170] and relates bilevel methods to other methods in the framework. Although not covered in the above framework, Section 7.4 also compares bilevel methods to a third general category: "single-level" hyperparameter learning methods. Like bilevel methods, single-level methods learn hyperparameters in a supervised manner. However, they generally learn parameters that sparsify the training images, $\{\boldsymbol{x}_j^{\text{true}}\}$, and do not use the noisy data, $\{\boldsymbol{y}_j\}$. This last comparison demonstrates the benefit of task-based approaches. Of course, there is variety among bilevel methods; this discussion is meant to provide perspective and general relations to increase understanding, rather than to narrow the definition or application of any method.

7.1 Connection: Learnable Optimization Algorithms

Learning parameters in unrolled optimization algorithms to create an inference-aided network, often called a Learnable Optimization Algorithm (LOA), is a quickly growing area of research [171]. The first such instance was a learned version of the Iterative Shrinkage and Thresholding Algorithm (ISTA), called LISTA [172]. Similar to the bilevel unrolling method, a LOA typically starts from a traditional, inference-based optimization algorithm, unrolls multiple iterations, and then learns parameters using end-to-end training.

There are many unrolled methods for image reconstruction [171]. Two examples that explicitly state the bilevel connection are [34], [173]; both set-up a bilevel problem with a DNN as a regularizer and then allow the parameters to vary by iteration, *i.e.*, learning $\boldsymbol{c}_k^{(t)}$ where t denotes the lower-level iteration. Ref. [173] motivated the use of an unrolled DNN over more inference-based methods by the lack of an accurate forward model, specifically coil sensitivity maps, for MRI reconstruction. Other examples of unrolled networks are [174], which unrolls the Field of Experts model [55] (see Sections 2.3 and 6.1 for how the Field of Experts model has inspired many bilevel methods); [175], which unrolls the convolutional analysis operator model [61] (see (2.12)); and [141], which discusses the connection to meta-learning.

Unlike the unrolled approach to bilevel learning described in Section 4.4, many LOAs depart from their base cost function and "only superficially resemble the steps of optimization algorithms" [34]. For example, unrolled algorithms may "untie" the gradient from the original cost function, *e.g.*, using $\tilde{\boldsymbol{A}}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$, instead of $\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$ for the gradient of the common 2-norm data-fit term, where $\tilde{\boldsymbol{A}}'$ is learned or otherwise differs from the adjoint of $\boldsymbol{A}$. LOAs that allow the learned parameters to vary every unrolled iteration or learn step size and momentum parameters further depart from a cost function perspective.

In addition to selecting which variables to learn, one must decide how many iterations to unroll for both bilevel unrolled approaches and LOAs. Most methods pick a set number of iterations in advance, perhaps based on previous experience, initial trials, or the available computational resources. Using a set number of iterations yields an algorithm with predictable run times and allows the learned parameters to adapt to the given number of iterations. Further, picking a small number of iterations can act as implicit regularization, comparable to early stopping in machine learning, which may be helpful when the amount of training data is small relative to the number of hyperparameters in the unrolled algorithm [141].

One can also use a convergence criteria to determine the number of iterations to evaluate, rather than selecting a number in advance [132]. This convergence-based method more closely follows classic inference-based optimization algorithms. A benefit of running the lower-level optimization algorithm until convergence is that one could switch optimization algorithms between training and testing, especially for strictly convex lower-level cost functions, and still expect the learned parameters to perform similarly. This ability to switch optimization algorithms means one could use faster, but not differentiable, algorithms at test-time, such as accelerated gradient descent methods with adaptive restart [148]. We are unaware of any bilevel methods that have exploited this possibility.

Even within the unrolling methodology, one must make several design decisions. To remain most closely tied to the original optimization algorithm, an unrolled method might fix a large number of iterations or run the optimization algorithm until convergence, use the same

parameters every layer, and calculate the step size based on the Lipschitz constant every upper-level iteration (see discussion in Section 4.4.1). Like all design decisions, there are trade-offs and the literature shows many successful methods that benefit from the increased generality of designing LOAs that are further removed from their cost function roots [171]. Echoing the ideas from [170], the design should be based on the specific application and relative availability, reliability, and importance of prior knowledge and training data.

This survey focuses on unrolled methods that are closely tied to the original bilevel formulation; [171] reviews LOAs more broadly. A benefit of maintaining the connection to the original cost function and optimization algorithm is that, once trained, the lower-level problem in an unrolled bilevel method inherits any theoretical and convergence results from the corresponding optimization method. The corresponding benefit for LOAs is increased flexibility in network architecture.

7.2 Connection: Equilibrium-based Networks

Equilibrium-based, or fixed point, networks are related to both LOAs and the minimizer approach from Section 4.2. The idea was proposed only recently in [176], but has received much attention. From the unrolled perspective, equilibrium networks consider what happens when the number of unrolled iterations approaches infinity. Alternatively, they can be viewed as a single, implicit layer; as in the minimizer approach, the output is the solution to a nonlinear equation.

We first consider the unrolled perspective. If an algorithm Ψ is a contraction, *i.e.*,

$$\|\Psi(\boldsymbol{x}_1\,;\,\boldsymbol{\gamma}) - \Psi(\boldsymbol{x}_2\,;\,\boldsymbol{\gamma})\| \leq \delta\,\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|\,,\ \forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{F}^N$$

for some parameter $\delta \in [0, 1)$, then the sequence of iterates will eventually converge to a fixed-point of Ψ. If the optimization algorithm optimizes a cost function with a data-fit and regularization term, then the equilibrium network approach is equivalent to a bilevel method. For a given value of $\boldsymbol{\gamma}$, the contraction condition is typically easy to satisfy by selecting an appropriate step-size in algorithms like gradient descent. Ref. [131] provides conditions on deep equilibrium models specific to

optimization algorithms based on gradient descent, proximal gradient descent, and ADMM that ensure convergence.

Re-using some of our bilevel notation, let $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ denote a fixed-point of an equilibrium network. The derivation for finding $\nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) \in \mathbb{F}^{N\times R}$ follows similar steps to the IFT perspective on the bilevel minimizer approach in Section 4.2.1. The key difference is that rather than using the first-order optimally condition as in the minimizer approach (4.3), the equilibrium method considers the lower-level minimizer to be a fixed point of an optimization algorithm.

When the goal of the lower level problem is to find a fixed point, the bilevel problem becomes

$$\underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \underbrace{\ell\left(\boldsymbol{\gamma}\,;\,\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\right)}_{\ell(\boldsymbol{\gamma})} \text{ s.t. } \underbrace{\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \Psi(\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\,;\,\boldsymbol{\gamma})}_{\text{Fixed point equation}}. \tag{7.1}$$

Similar to the IFT perspective, one can differentiate both sides of the fixed point equation using the chain rule

$$\nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \left(\nabla_{\boldsymbol{x}}\Psi(\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\,;\,\boldsymbol{\gamma})\right)\nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) + \nabla_{\boldsymbol{\gamma}}\Psi(\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\,;\,\boldsymbol{\gamma})$$

and then rearrange to derive an expression for $\nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$

$$\nabla_{\boldsymbol{\gamma}}\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \left(\boldsymbol{I} - \underbrace{\left(\nabla_{\boldsymbol{x}}\Psi(\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\,;\,\boldsymbol{\gamma})\right)}_{\hat{\boldsymbol{J}}}\right)^{-1}\nabla_{\boldsymbol{\gamma}}\Psi(\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\,;\,\boldsymbol{\gamma}). \tag{7.2}$$

The matrix $\hat{\boldsymbol{J}}$ is the Jacobian of the optimization algorithm, evaluated at the fixed point $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$.

Substituting (7.2) into the expression for the upper-level gradient (4.2) yields

$$\nabla\ell(\boldsymbol{\gamma}) = \nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\hat{\boldsymbol{x}}(\boldsymbol{\gamma})) + \left(\nabla_{\boldsymbol{\gamma}}\Psi(\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\,;\,\boldsymbol{\gamma})\right)'\left(\boldsymbol{I} - \hat{\boldsymbol{J}}\right)^{-1}\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\hat{\boldsymbol{x}}(\boldsymbol{\gamma})). \tag{7.3}$$

If the optimization is standard gradient descent, *i.e.*, $\Psi(\boldsymbol{x}\,;\boldsymbol{\gamma}) = \boldsymbol{x} - \alpha_\Phi\nabla_{\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})$, then

$$\begin{aligned}\nabla_{\boldsymbol{\gamma}}\Psi(\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\,;\,\boldsymbol{\gamma}) &= \text{-}\alpha_\Phi\nabla_{\boldsymbol{x}\boldsymbol{\gamma}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}) \text{ and}\\ \nabla_{\boldsymbol{x}}\Psi(\hat{\boldsymbol{x}}(\boldsymbol{\gamma})\,;\,\boldsymbol{\gamma}) &= \boldsymbol{I} - \alpha_\Phi\nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}).\end{aligned}$$

Substituting these expressions into (7.2) yields the gradient as derived using the IFT perspective in the minimizer approach (4.5), showing the close connection between the equilibrium and minimizer approach.

Similar to the minimizer approach, one can use any algorithm to find a fixed point $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ of Ψ. For example, [176] used a quasi-Newton method and [131] used a standard fixed-point accelerated method. One can use any fixed point algorithm to find $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$; the algorithm used need correspond to Ψ in (7.1); for example, Ψ could be standard gradient descent, even if one uses a more advanced algorithm to initially compute $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$. Another similarity to the minimizer approach is that the learned parameters are optimal at convergence of the lower-level problem, rather than after a fixed number of lower-level iterations. Therefore, the end-user can trade-off accuracy and compute requirements at test time, unlike in unrolled approaches where the number of iterations is pre-decided.

Although the equilibrium model is the limit as the number of unrolled iterations approaches infinity, computing $\nabla \ell(\boldsymbol{\gamma})$ does not require backpropagation nor storing any intermediate matrices. The trade-off is that (7.3) requires multiplying $(\boldsymbol{I} - \hat{\boldsymbol{J}})^{-1}$ by a vector. The remaining computations in the full upper-level gradient (7.3) are straightforward. Similar to the required Hessian inverse-vector product in the minimizer approach, one can use an iterative algorithm to approximate the matrix inverse. Ref. [131] notes that the inverse matrix-vector product

$$\boldsymbol{v} = (\boldsymbol{I} - \hat{\boldsymbol{J}})^{-1} \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})),$$

is a fixed point of the equation

$$\boldsymbol{v} = \hat{\boldsymbol{J}}\boldsymbol{v} + \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \hat{\boldsymbol{x}}(\boldsymbol{\gamma})).$$

Therefore, one can use any fixed-point solver to compute the matrix-vector product. Another way to decrease the computational cost of the Jacobian product is to use the method from [122]: if a quasi-Newton algorithm is used to estimate the Jacobian for the forward step of computing $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$, then one can "re-use" this estimated Jacobian to find $\nabla \ell(\boldsymbol{\gamma})$.

Fixed point networks can also be viewed from the perspective of unrolled methods. Although it is often infeasible to backpropagate through

the large number of iterations required to reach a fixed point, backpropagating through the last few iterations yields a valid gradient estimate for $\nabla_{\boldsymbol{\gamma}} \hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ [137]. Ref. [137] proves that this "truncated backpropagation" approach converges to a stationary point of the upper-level loss when the lower-level cost function is locally strongly convex around $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ because the backpropagation gradient error decays exponentially with reverse depth. A similar approach is to use $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ at every backpropagation step rather than previous iterates. Ref. [177] shows this is equivalent to approximating the matrix inverse in the minimizer approach using a Neumann series.

Recently, [178] proposed a Jacobian-free method to find $\nabla \ell(\boldsymbol{\gamma})$ that takes the approach from [137] to the extreme case: it considers unrolling a single layer. The approach in [178] is equivalent to viewing the deep equilibrium network as a single layer network where the initialization is the fixed-point, *i.e.*, using $\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \Psi(\boldsymbol{x}^{(0)}\,;\boldsymbol{\gamma})$ in the unrolled method with $\boldsymbol{x}^{(0)} = \hat{\boldsymbol{x}}(\boldsymbol{\gamma})$. With this new perspective, it is easy to use existing backpropagation tools to compute the derivative through the single layer network. Assuming that the network is Lipschitz, contractive, and differentiable and that the upper-level loss is differentiable, [178] shows the Jacobian-free gradient is a descent direction for estimates of $\hat{\boldsymbol{x}}(\boldsymbol{\gamma})$ that are within some error bound of the true fixed point.

Deep equilibrium networks can be fully learned or they can incorporate physics-based models into their network architecture and move into the inference-aided networks category in Fig. 7.1. For example, [131], [179] incorporated system matrices into fixed point networks and applied them to MRI and CT image reconstruction problems.

7.3 Connection: Plug-and-play Priors

The Plug-and-Play (PNP) framework [180] is an example of a DNN-aided inference method. It is similar to bilevel methods in its dependence on the forward model. However, unlike bilevel methods, the PNP framework need not be connected to a specific lower-level cost function and it leverages pre-trained denoisers rather than training them for a specific task.

As a brief overview of the PNP framework, consider rewriting the

generic data-fit plus regularizer optimization problem (2.1) with an auxiliary variable:

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in \mathbb{F}^N}{\operatorname{argmin}} \underbrace{\overbrace{d(\boldsymbol{x}\,;\boldsymbol{y})}^{\text{Data-fit}} + \beta \overbrace{R(\boldsymbol{z}\,;\boldsymbol{\gamma})}^{\text{Regularizer}}}_{\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})} \quad \text{s.t. } \boldsymbol{x} = \boldsymbol{z}. \tag{7.4}$$

Using ADMM [181] to solve this constrained optimization problem and rearranging variables yields the following iterative optimization approach for (7.4):

$$\begin{aligned}
\boldsymbol{x}^{(u+1)} &= \underset{\boldsymbol{x}}{\operatorname{argmin}}\, d(\boldsymbol{x}\,;\boldsymbol{y}) + \frac{\lambda}{2}\|\boldsymbol{x} - \underbrace{(\boldsymbol{z}^{(u)} - \boldsymbol{u}^{(u)})}_{\tilde{\boldsymbol{x}}}\|_2^2 && = \operatorname{prox}_{\frac{1}{\lambda} d(\boldsymbol{x}\,;\boldsymbol{y})}(\tilde{\boldsymbol{x}}) \\
\boldsymbol{z}^{(u)} &= \underset{\boldsymbol{z}}{\operatorname{argmin}}\, \beta R(\boldsymbol{z}\,;\boldsymbol{\gamma}) + \frac{\lambda}{2}\|\boldsymbol{z} - \underbrace{(\boldsymbol{x}^{(u)} + \boldsymbol{u}^{(u)})}_{\tilde{\boldsymbol{z}}}\|_2^2 && = \operatorname{prox}_{\frac{\beta}{\lambda} R(\boldsymbol{z}\,;\boldsymbol{\gamma})}(\tilde{\boldsymbol{z}}) \\
\boldsymbol{u}^{(u+1)} &= \boldsymbol{u}^{(u)} + (\boldsymbol{x}^{(u+1)} - \boldsymbol{z}^{(u+1)}),
\end{aligned}$$

where λ is an ADMM penalty parameter that effects the convergence rate (but not the limit, for convex problems). The first step is a proximal update for $\boldsymbol{x}$ that uses the forward model but does not depend on the regularizer. Conversely, the second step is proximal update for the split variable $\boldsymbol{z}$ that depends on the regularizer, but is agnostic of the forward model. This step acts as a denoiser. The final step is the dual variable update and encourages $\boldsymbol{x}^{(u)} \approx \boldsymbol{z}^{(u)}$ as $u \to \infty$.

The key insight from [180] is that the above update equations separate the forward model and denoiser. Thus, one can substitute, or "plug in," a wide range of denoisers for the $\boldsymbol{z}$ update, in place of its proximal update, while keeping the data-fit update independent.

Whereas in the original ADMM approach, the parameter λ has no effect on the final image for convex cost functions, in the PNP framework that parameter does affect image quality. Thus, one could also use training data to tune the λ in a bilevel manner. Although PNP allows one to substitute a pre-trained denoiser, one could additionally tune the parameters in the denoiser. Ref. [182] provides one such example of starting from a PNP framework then learning denoising parameters and λ that vary by iteration.

A large motivation for the PNP framework is the abundance of advanced denoising methods, including ones that are not associated with an optimization problem such as BM3D [162]. However, using existing denoisers sacrifices the ability to learn parameters to work well with the specific forward model, as is done in task-based methods. As simple examples of how learned parameters may differ when $\boldsymbol{A}$ changes, [68] found that different filters worked better for image denoising versus image inpainting and [25] found that unrolled deblurring methods required more upper-level iterations than unrolled denoising methods. A more complicated example is using bilevel methods to learn some aspect of $\boldsymbol{A}$ alongside some aspect of the regularizer, *e.g.*, [28] learned a sparse sampling matrix and tuning parameter for MRI that are adaptive to the regularization for the image reconstruction problem.

7.4 Connection: Single-Level Parameter Learning

Section 2.3 briefly discussed some approaches to learning analysis operators. This section further motivates the task-based bilevel set-up by discussing the filter learning constraints imposed in single-level hyperparameter learning methods.

As summarized in Section 2.3, the earliest methods for learning analysis regularizers had no constraints on the analysis operators. Those approaches learned filters from training data to make a prior distribution match the observed data distribution. In contrast, more recent approaches to filter learning minimize a cost function that requires either a penalty function or constraint on the operators to ensure filter diversity. For reference, the cost functions mentioned in Section 2.3 were:

$$\text{AOL}: \underset{\boldsymbol{\Omega},\, \boldsymbol{X}}{\operatorname{argmin}} \left\|\boldsymbol{\Omega}\boldsymbol{X}\right\|_1 + \frac{\beta}{2}\left\|\boldsymbol{Y}-\boldsymbol{X}\right\|^2 \text{ s.t. } \boldsymbol{\Omega} \in \mathcal{S},$$

$$\text{TL}: \underset{\boldsymbol{\Omega}\in\mathbb{F}^{S\times S},\, \boldsymbol{X}}{\operatorname{argmin}} \left\|\boldsymbol{\Omega}\boldsymbol{Y}-\boldsymbol{X}\right\|_2^2 + R(\boldsymbol{\Omega}) \text{ s.t. } \left\|\boldsymbol{X}_i\right\|_0 \leq \alpha \; \forall i,$$

$$\text{CAOL}: \underset{[\boldsymbol{c}_1,\ldots,\boldsymbol{c}_K]}{\operatorname{argmin}} \min_{\boldsymbol{z}} \sum_{k=1}^{K} \frac{1}{2}\left\|\boldsymbol{c}_k \circledast \boldsymbol{x} - \boldsymbol{z}\right\|_2^2 + \beta\left\|\boldsymbol{z}_k\right\|_0 \text{ s.t. } [\boldsymbol{c}_1,\ldots,\boldsymbol{c}_K] \in \mathcal{S},$$

where AOL is analysis operator learning [59], TL is transform learning

[51], and CAOL is convolutional analysis operator learning [61]. In the following discussion of constraint sets, the equivalent filter matrix for CAOL has the convolutional kernels as rows:

$$\boldsymbol{\Omega}_{\text{CAOL}} = \begin{bmatrix} \boldsymbol{c}_1' \\ \vdots \\ \boldsymbol{c}_K' \end{bmatrix}.$$

While there are many other proposed cost functions in the literature, using different norms or including additional variables, these three examples capture the most common structures for filter learning.

In all the above cost functions, if one removed the constraint or regularizer, then the trivial solution would be to learn zero filters for $\boldsymbol{\Omega}$. Furthermore, a simple row norm constraint on $\boldsymbol{\Omega}$ would be insufficient, as then the minimizer would contain a single filter that is repeated many times. (In contrast, a unit norm constraint typically suffices for dictionary learning.) A row norm constraint plus a full rank constraint is also insufficient because $\boldsymbol{\Omega}$ can have full rank while being arbitrarily close to the rank-1 case of having a single repeated row.

The choice of constraint set $\mathcal{S}$ is important in single-level learning. Many methods constrain analysis operators to satisfy a tight frame constraint. A matrix $\boldsymbol{A}$ is a tight frame if there is a positive constant, α, such that

$$\left\|\boldsymbol{A}'\boldsymbol{x}\right\|_2^2 = \sum_i |\langle \boldsymbol{q}_i, \boldsymbol{x}\rangle|^2 = \alpha \left\|\boldsymbol{x}\right\|_2^2, \ \forall \boldsymbol{x}$$

where $\boldsymbol{q}_i$ is the ith column of $\boldsymbol{A}$. This tight frame condition is equivalent to $\boldsymbol{A}\boldsymbol{A}' = \alpha \boldsymbol{I}$ for some positive constant α. Most analysis operators are defined with filters in their rows, so a tight frame requirement on the filters appears as the constraint $\boldsymbol{\Omega}'\boldsymbol{\Omega} = \alpha \boldsymbol{I}$.

Under the tight frame constraint for the filters, $\boldsymbol{\Omega}$ must be square or tall, so the filters are complete or over-complete. However, [59] found that the frame constraint was insufficient when learning over-complete operators, as the "excess" rows past full-rank tended to be all zeros. Therefore, [59] imposed a uniformly-normalized tight frame constraint: each row of the $\boldsymbol{\Omega}$ had to have unit norm and the filters had to form a tight frame.

Ref. [50] similarly constrained $\mathbf{\Omega}$ to have unit-norm rows with the filters forming a frame (though not tight). Such loosening of the tight frame constraint to a frame constraint could lead to the problem of learning almost identical rows, as discussed above. To prevent this issue, [50] additionally included a penalty that encourages distinct rows:

$$-\sum_{k}\sum_{\tilde{k}<k}\log\left(1-(\boldsymbol{\omega}_{\tilde{k}}'\boldsymbol{\omega}_k)^2\right). \tag{7.5}$$

One possible concern with a tight frame constraint is that it requires the filters to span all of $\mathbb{F}^N$, so every spatial frequency can pass through at least one filter. However, most images are not zero-mean and have piece-wise constant regions, so the zero frequency component is not sparse. Ref. [59] modified the tight-frame constraint to require $\mathbf{\Omega}$ to span some space (*e.g.*, the space orthogonal to the zero frequency term). Likewise, [183] extended the CAOL algorithm to include handcrafted filters, such as a zero frequency term, that can then be used or discarded when reconstructing images. In the bilevel literature, [30], [31] similarly ensured that learned filters had no zero frequency component by learning coefficients for a linear combination of filter basis vectors, rather than learning the filters directly; see Section 6.1.

As an alternative to imposing a strict constraint on the filters, one can penalize $\mathbf{\Omega}$ to encourage filter diversity, as in (7.5). Using a penalty has the advantage of being able to learn any size (under- or over-complete) $\mathbf{\Omega}$ and not *requiring* the filters to represent all frequencies. For example, as an alternative to the tight frame constraint, [61] proposed a version of CAOL using the following regularizer (to within scaling constants)

$$R(\mathbf{\Omega}) = \beta \left\| \mathbf{\Omega}\mathbf{\Omega}' - \boldsymbol{I} \right\|^2$$

and a unit norm constraint on the filters. Ref. [53] included a similar penalty to (7.5), but with the inner product being divided by the norm of the filters as the filters were not constrained to unit norm. All such variations on this penalty are to encourage filter diversity.

To ensure a square $\mathbf{\Omega}$ is full rank, while also encouraging it to be well-conditioned, [51] used a regularizer that includes a term of the form

$$R(\mathbf{\Omega}) = \text{-}\beta_1 \log\left(|\mathbf{\Omega}|\right).$$

The log determinant term is known as a log barrier; it forces $\mathbf{\Omega}$ to have full rank because of the asymptote of the log function. Ref. [53] includes a similar log barrier regularization term in terms of the eigenvalues of $\mathbf{\Omega}$ to ensure it is left-invertible.

As another example of a filter penalty regularizer, both [51] and [53], include the following regularization term

$$R(\mathbf{\Omega}) = \beta_2 \|\mathbf{\Omega}\|_F^2 ,$$

rather than constraining the norm of the filters. This Frobenius norm addresses the scale ambiguity in the analysis and transform formulations and ensures the filter coefficients do not grow too large in magnitude.

Yet another approach to encouraging filter diversity is to consider the frequency response of the set of filters. Pfister and Bresler [53] discuss different constraint options for filter banks based on convolution strides to ensure perfect reconstruction. When the stride is one and one considers circular boundary conditions, the filters can perfectly reconstruct any signal as long as they pass the N discrete Fourier transform frequencies. Tight frames satisfy this constraint, but the constraint is more relaxed than a tight frame constraint.

Section 6 discussed some (relatively rare) bilevel problems with penalties on the learned hyperparameters, but, notably, there are no constraints nor penalties on the filters in the bilevel method (Ex)! Because of its task-based nature, filters learned via the bilevel method should be those that are best for image reconstruction. Thus, one should not have to worry about redundant filters, zero filters, or filters with excessively large coefficients. This property is one of the key benefits of bilevel methods.

7.5 Future Directions

Throughout this review, we mentioned a few areas for future work on bilevel methods. This section highlights some of the avenues that we think are particularly promising.

Advancing upper-level loss function design is identified as future work in many bilevel papers. Despite the abundance of research on image quality metrics (see Section 3.1), most bilevel methods use squared error

for the upper-level loss function (see Section 6.2 for exceptions). Using loss functions that better match the end-application of the images is a clear future direction for bilevel methods that nicely aligns with their task-based nature. For example, in the medical imaging field there is a large literature on objective measures of image quality [184], often based on mathematical observers designed to emulate human performance on signal detection tasks, *e.g.*, in situations where a lesion's location is unknown [185]. To our knowledge, there has been little if any work to date on using such mathematical observers to define loss functions for bilevel methods or for training CNN models, though there has been work on CNN-based observers [186]. Using task-based metrics for bilevel methods and CNN training is a natural direction for future work that could bridge the extensive literature on such metrics with the image reconstruction field.

Unsupervised bilevel problems are exceptions to the trend of using squared error for the upper-level loss function. Section 6.2 considered a few unsupervised bilevel methods that use noise statistics to estimate the quality of the reconstructed images, *e.g.*, [142], [143], [146] [93], [94]. One extension to the unsupervised setting is the semi-supervised setting, where one might have access to a few clean training samples and additional, noisy training samples.

A related opportunity for future work is to use bilevel methods to learn patient-adaptive parameters. The population-based learning approach considered in (1.5) learns hyperparameters that are best *on average* over the set of training images. In contrast, a patient-adaptive approach tunes hyperparameters for every input image. For example, one could learn filters and initial tuning parameters offline from a training dataset and then adjust the tuning parameters when reconstructing a specific image, *e.g.*, using approaches such as the unsupervised approaches in Section 6.2. An alternative approach for adapting hyperparameters at test time is to learn a mapping from the input data to the set of hyperparameters [54], [187].

Just as considering more advanced image quality metrics for the upper-level loss function is a promising area for future work, bilevel methods can likely be improved by using more advanced lower-level cost functions. For example, one could use bilevel methods to learn

multi-scale filters, which can increase the receptive field of a regularizer and provide a more natural representation for data that is inherently multiscale [188], [189]. Perhaps due to the already challenging and non-convex nature of bilevel problems, most methods consider relatively simple convex lower-level cost functions. Papers that examine non-convex regularizers, *e.g.*, [30], [60], conclude that non-convex regularizers lead to more accurate image reconstructions, likely due to better matching the statistics of natural images. This observation aligns with the simple denoising experimental results in [190], where learned filters with (CR1N) as the regularizer yielded noisier signals than signals denoised with a hand-crafted filter with the non-convex 0-norm regularizer. In other words, the structure of the regularizer matters in addition to how one learns the filters.

In addition to non-convexity, future bilevel methods could consider non-smooth cost functions. Many bilevel methods require the lower-level cost to be smooth. Exceptions include the translation to a single level approach (Section 4.3), which uses the 1-norm as the lower-level regularizer, and unrolled methods, which can be applied to non-smooth cost functions as long as the optimization algorithm has smooth updates (Section 4.4.2). The impact of smoothing the cost function on the perceptual quality of the reconstructed image is largely unknown.

Another avenue for future work is based on the fact that $\boldsymbol{x}^{\text{true}}$ is really a continuous-space function. A few methods, *e.g.*, [11], [12], develop bilevel methods in continuous-space. However, the majority of methods use discretized forward models without considering the impact of this simplification (as done in this review paper). Future investigations of bilevel methods should strive to avoid the "inverse crime" [191] implicit in (1.4) where the data is synthesized using the same discretization assumed by the reconstruction method.

Future work may also consider how to more closely tie the bilevel method to a statistical modeling framework and leverage progress made in that field. Many bilevel methods for filter learning use the Field of Experts [55] as a starting point. Ref. [55] takes a maximum-likelihood perspective and learns parameters to model the training data distribution. In contrast, bilevel methods such as (Ex) have their roots in a maximum *a posteriori* perspective. While this approach is

motivated by and aligns with the task-based nature of bilevel methods [31], it is not clear how well the learned parameters reflect a prior or how to use the learned parameters to generate model uncertainties. Ideas from the Bayesian statistics literature, such as Monte Carlo methods, may be a promising avenue for future research.

Related to connecting bilevel methods and statistical processes, an interesting opportunity for a stochastic bilevel formulation is to add different noise realizations in (1.4), providing an uncountable ensemble of $(\boldsymbol{x}, \boldsymbol{y})$ training tuples, where the expectation in (1.5) is over the distribution of noise realizations. Yet another possibility is to have a truly random set of training images $\boldsymbol{x}^{\text{true}}$ drawn from some distribution. For example, [192] trained a CNN-based CT reconstruction method using an ensemble of images consisting of randomly generated ellipses. Other variations, such as random rotations or warps, have also been used for data augmentation [193]. One could combine such a random ensemble of images with a random ensemble of noise realizations, in which case the expectation in (1.5) would be taken over both the image and noise distributions. We are unaware of any bilevel methods for imaging that exploit this full generality. Future literature on stochastic methods should clearly state what expectation is used and may consider exploiting a more general definition of randomness.

7.6 Summary of Advantages and Disadvantages

Like the methods described in [170], bilevel methods for computational imaging involve mixing inference-based optimization approaches with learning-based approaches to leverage benefits of both techniques.

Inference-based approaches use prior knowledge, usually in the form of a forward model and an object model, to reconstruct images. Typically the forward model, $\boldsymbol{A}$, is under-determined, so some form of regularization based on the object model is essential. Regularizers always involve some number of adjustable parameters; traditionally inference-based methods select such parameters empirically or using basic image properties like resolution and noise [112], [194]. The regularization parameters may also be learned from training to maximize SNR [195] or detection task performance [196] in a bilevel manner (often using a

grid or random search due to the relatively small number of learnable parameters). When the forward model and object model are well-known and easy to incorporate in a cost function, inference-based methods can yield accurate reconstructions without the need for large datasets of clean training data.

Learning-based approaches use training datasets to learn a prior. Recently, learning-based approaches have achieved remarkable reconstruction accuracy in practice, largely due to the increased availability in computational resources and larger, more accessible training datasets [4], [5]. However, many (deep) learning methods lack theoretical guarantees and explainability and finding sufficient training data is still challenging in many applications. Both of these challenges may impede adoption of learning-based methods in clinical practice for some applications, such as medical image reconstruction [197]. Some deep learning methods for CT image reconstruction were approved for clinical use in 2019 [198]; early studies have shown such methods can significantly reduce noise but may also compromise low-contrast spatial resolution [199].

Combining inference-based and learning-based approaches allows the integration of learning from training data while using smaller training datasets by incorporating prior knowledge. Such mixed methods often maintain interpretability from the inference-based roots while using learning to provide adaptive regularization. Thus, the benefits of bilevel methods in this review's introduction are generally shared among the methods described in [170]: theoretical guarantees, competitive performance in terms of reconstruction accuracy, and similar performance to learned networks with a fraction of the free parameters, *e.g.*, [29], [34].

What distinguishes bilevel methods from the other methods in the inference-based to learning-based spectrum in Fig. 7.1? While one can argue that the conventional CNN and deep learning approach is always bilevel in the sense that the hyperparameters are trained to minimize a loss function, this review considered bilevel methods with the cost function structure (LL). The regularization term in (LL) could be based on a DNN [34], but we followed the bilevel literature that focuses on priors/regularizers, such as in (Ex), maintaining a stronger connection to traditional cost function design.

Another lens for understanding bilevel methods is extending single-

level hyperparameter optimization approaches to be task-based, bilevel approaches. Single-level approaches to image reconstruction, such as those using dictionary learning [80], convolutional analysis operator learning [61], and convolutional dictionary learning [200], [201], generally aim to learn characteristics of a training dataset, with the idea that these characteristics can then be used in a prior for an image reconstruction task. While such an approach may learn more general information, [124], [190] showed that a common single-level optimization strategy resulted in learning a regularizer that was suboptimal for the simple task of image denoising.

As further evidence of the benefit of task-based learning, [124] found that the lack of constraints in the bilevel filter learning problem is important; the learned filters used the flexibility of the model and were not orthonormal, whereas orthonormality is a constraint often imposed in single-level models (see Section 7.4). Ref. [60] showed how the task-based nature adapts to training data; total variation based regularization works well for piece-wise constant images but less so for natural images. Beyond adapting to the training dataset, bilevel methods are task-based in terms of adapting to the level of noise; [27] found the learned tuning parameters for image denoising go to 0 as the noise goes to 0, since no regularization is needed in the absence of noise for well-determined problems.

A primary disadvantage cited for most bilevel methods is the computational cost compared to single-level hyperparameter optimization methods or other methods with a smaller learning component. In turn, the main driver behind the large computational cost of gradient descent based bilevel optimization methods is that one typically has to optimize the lower-level cost function many times, either to some tolerance or for a certain number of iterations. The computational cost involves a trade-off because how accurately one optimizes the lower-level problem can impact the quality of the learned parameters. For example, [30], [60] both claim better denoising accuracy than [31] because they optimize the lower-level problem more accurately. Similarly, [124] notes that learning will fail if the lower-level cost is not optimized to sufficient accuracy.

There are various strategies to decrease the computational cost for

bilevel methods. Some are relatively intuitive and applicable to a wide range of problems in machine learning. For example, [124] used larger batch size as the iterations continue, [11] increased the batch size if a gradient step in $\boldsymbol{\gamma}$ does not sufficiently improve the loss function, and [27] tightened the accuracy requirement for the gradient estimation over iterations. These strategies all save computation by starting with rougher approximations near the beginning of the optimization method, when $\boldsymbol{\gamma}^{(u)}$ is likely far from $\hat{\boldsymbol{\gamma}}$, while using a relatively accurate solution by the end of the algorithm.

Another disadvantage of bilevel methods is that, while the optimization algorithm for the lower-level problem often has theoretical convergence guarantees, and the lower-level cost is often designed to be strictly convex, the full bilevel problem (UL) is usually non-convex, so the quality of the learned hyperparameters can depend on initialization. Thus, in practice, one requires a strategy for initializing $\boldsymbol{\gamma}$. For example, for (Ex), one may decide to use a single-level filter learning technique such as the Field of Experts [55] to initialize the hyperparameters. Or, one can use a handcrafted set of filters, such as the DCT filters (or a subset thereof). Other hyperparameters often have similar warm start options. Despite the non-convexity, papers that tested multiple initializations generally found similarly good solutions surprisingly often, *e.g.*, [27], [30], [142].

There is no one correct answer for how much a method should use prior information or learning techniques, and it is unlikely that any single approach can be the best for all image reconstruction applications. Like most engineering problems, the trade-off is application-dependent. One should (minimally) consider the amount of training data available, how representative the training data is of the test data, how underdetermined the forward model is (*i.e.*, how strong of regularization is needed), how well-known the object model is, the importance of theoretical guarantees and explainability, and the available computational resources at training time and at test time. Bilevel methods show particular promise for applications where training data is limited and/or explainability is highly valued, such as in medical imaging.

Acknowledgements

This work was made possible in part due to the support of NIH grant R01 EB023618, NSF grant IIS 1838179, and the Rackham Predoctoral Fellowship. The authors would like to thank Lindon Roberts for helpful email discussion of [27], Mike McCann for general discussion of bilevel approaches and of [125] specifically, and Avrajit Ghosh and Saiprasad Ravishankar for discussion of [125]. The authors would also like to thank Qing Qu and the anonymous reviewers, whose suggestions were a great help in strengthening and clarifying this review.

Appendices

A

Background: Primal-Dual Formulations

This appendix briefly reviews primal-dual analysis as it applies to (Ex). Section 3.3 in [41] provides a more general but brief introduction to the notion of conjugate functions and duality and [202] goes into more depth on duality.

The conjugate of a function $f : \mathbb{R}^N \to \mathbb{R} \cup \{\text{-}\infty, \infty\}$ is denoted $f^* : \mathbb{R}^N \to \mathbb{R} \cup \{\text{-}\infty, \infty\}$, and is defined as

$$f^*(\boldsymbol{d}) = \sup_{\boldsymbol{x} \in \operatorname{domain}(f)} \boldsymbol{d}'\boldsymbol{x} - f(\boldsymbol{x}), \tag{A.1}$$

where $\boldsymbol{d} \in \mathbb{R}^N$ is a dual variable. The derivations below use the following two conjugate function relations.

1. When $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2$ for $\boldsymbol{y} \in \mathbb{R}^N$, the conjugate function is

$$f^*(\boldsymbol{d}) = \sup_{\boldsymbol{x} \in \mathbb{R}^N} \boldsymbol{d}'\boldsymbol{x} - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2.$$

The maximizer of the quadratic cost function f^* is

$$\hat{\boldsymbol{x}} = \boldsymbol{y} + \boldsymbol{d} \tag{A.2}$$

and the maximum value simplifies to

$$f^*(\boldsymbol{d}) = \frac{1}{2}\|\boldsymbol{d} + \boldsymbol{y}\|^2 - \frac{1}{2}\|\boldsymbol{y}\|^2. \tag{A.3}$$

2. When $\phi(z) = |z|$ is defined on $\mathbb{R}$, the conjugate function is

$$\phi^*(d) = \sup_{z \in \mathbb{R}} dz - |z|.$$

One can verify that the conjugate is

$$\phi^*(d) = \begin{cases} 0 & \text{if } |d| \leq 1 \\ \infty & \text{else} \end{cases} \tag{A.4}$$

and the corresponding sets of suprema are

$$\operatorname*{argmax}_{z \in \mathbb{R}} dz - |z| = \begin{cases} \text{sign}(d) \cdot \infty & \text{if } |d| > 1 \\ 0 & \text{if } |d| < 1 \\ [0, \infty) & \text{if } d = 1 \\ (\text{-}\infty, 0] & \text{if } d = \text{-}1. \end{cases} \tag{A.5}$$

Generalizing (A.4) to a vector, the conjugate function of the 1-norm is a characteristic function that is infinity if any element of the input vector is larger than 1 in absolute value.

Ref. [202, p. 50] provides a table with many more conjugate functions.

The biconjugate, denoted f^{**}, is the conjugate of f^*, *i.e.*,

$$f^{**}(\boldsymbol{x}) = \sup_{\boldsymbol{d} \in \text{domain}(f^*)} \boldsymbol{x}'\boldsymbol{d} - f^*(\boldsymbol{d}), \tag{A.6}$$

and is the largest convex, lower semi-continuous function below f. When f is convex and lower semi-continuous, the biconjugate is equal to the original function, *i.e.*, $f^{**} = f$. One can use the equality of the original function and the biconjugate to derive the saddle point and dual problems when f is convex.

Consider the specific lower-level problem with an analysis-based regularizer

$$\operatorname*{argmin}_{\boldsymbol{x} \in \mathbb{R}^N} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \mathbf{1}'\phi_\cdot(\boldsymbol{\Omega}\boldsymbol{x}), \tag{A.7}$$

where $\boldsymbol{\Omega} \in \mathbb{R}^{F \times N}$. When ϕ is convex, the corresponding saddle-point problem is

$$\operatorname*{argmin}_{\boldsymbol{x} \in \mathbb{R}^N} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \underbrace{\sup_{\boldsymbol{d} \in \mathbb{R}^F} \langle \boldsymbol{d}, \boldsymbol{\Omega}\boldsymbol{x} \rangle - \mathbf{1}'\phi^*_\cdot(\boldsymbol{d})}_{\mathbf{1}'\phi^{**}_\cdot(\boldsymbol{\Omega}\boldsymbol{x})},$$

where $\langle \cdot, \cdot, \rangle$ is the standard inner product. Under very mild conditions (satisfied for the absolute value function) [41], one can swap the minimum and supremum operations and write the **saddle-point problem** as

$$\sup_{\boldsymbol{d} \in \mathbb{R}^F} \min_{\boldsymbol{x} \in \mathbb{R}^N} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \langle \boldsymbol{d}, \boldsymbol{\Omega}\boldsymbol{x} \rangle - \mathbf{1}' \phi^*.(\boldsymbol{d}).$$

Substituting the conjugate of the 1-norm (A.4), the saddle-point problem is thus

$$\min_{\boldsymbol{x} \in \mathbb{R}^N} \min_{\boldsymbol{d} \in \mathbb{R}^F} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 - \langle \boldsymbol{d}, \boldsymbol{\Omega}\boldsymbol{x} \rangle \text{ s.t. } |d_i| \leq 1 \; \forall i. \tag{A.8}$$

We hereafter assume $\boldsymbol{A} = \boldsymbol{I}$ to derive the dual problem from the saddle-point problem. By grouping terms and re-arranging negative signs, the dual problem can be derived from the saddle point problem. For a general ϕ, the saddle-point problem is equivalent to

$$\begin{aligned} &\max_{\boldsymbol{d} \in \mathbb{R}^F} \text{-}\mathbf{1}' \phi^*.(\boldsymbol{d}) + \left(\min_{\boldsymbol{x} \in \mathbb{R}^N} \langle \boldsymbol{d}, \boldsymbol{\Omega}\boldsymbol{x} \rangle + \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{y}\|^2 \right) \\ &= \max_{\boldsymbol{d} \in \mathbb{R}^F} \text{-}\mathbf{1}' \phi^*.(\boldsymbol{d}) - \underbrace{\left(\max_{\boldsymbol{x} \in \mathbb{R}^N} \langle \text{-}\boldsymbol{\Omega}'\boldsymbol{d}, \boldsymbol{x} \rangle - \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{y}\|^2 \right)}_{f^*(\text{-}\boldsymbol{\Omega}'\boldsymbol{d})}, \end{aligned}$$

where the last line follows from properties of inner products. The expression in parenthesis is the conjugate function for the data-fit term, given in (A.3). Therefore, the dual problem for a general, convex ϕ is

$$\max_{\boldsymbol{d} \in \mathbb{R}^F} \text{-}\mathbf{1}' \phi^*.(\boldsymbol{d}) - f^*(\text{-}\boldsymbol{\Omega}'\boldsymbol{d}) = \text{-} \min_{\boldsymbol{d} \in \mathbb{R}^F} \mathbf{1}' \phi^*.(\boldsymbol{d}) + f^*(\text{-}\boldsymbol{\Omega}'\boldsymbol{d}).$$

Substituting the conjugates for the data-fit term (A.3) and the conjugate for the 1-norm regularizer (A.4), the **dual problem** for (A.7) with $\phi(z) = |z|$ becomes

$$\min_{\boldsymbol{d} \in \mathbb{R}^F} \frac{1}{2} \left\| \text{-}\boldsymbol{\Omega}'\boldsymbol{d} + \boldsymbol{y} \right\|^2 - \frac{1}{2} \|\boldsymbol{y}\|^2 \text{ s.t. } |d_i| \leq 1 \; \forall i. \tag{A.9}$$

When we require only the minimizer (not the minimum), an equivalent dual problem is

$$\hat{\boldsymbol{d}} = \operatorname*{argmin}_{\boldsymbol{d} \in \mathbb{R}^F} \frac{1}{2} \left\| \text{-}\boldsymbol{\Omega}'\boldsymbol{d} + \boldsymbol{y} \right\|^2 \text{ s.t. } |d_i| \leq 1 \; \forall i. \tag{A.10}$$

This dual problem is a constrained least squares problem and can be solved with a projected gradient descent method, optionally with momentum [148]. From (A.2), the primal minimizer can be recovered from the dual minimizer by

$$\hat{\boldsymbol{x}} = \boldsymbol{y} - \boldsymbol{\Omega}'\hat{\boldsymbol{d}}. \tag{A.11}$$

Finally, from (A.5), the dual variable is related to the filtered signal by

$$d_i \in \begin{cases} 1 & \text{if } [\boldsymbol{\Omega}\hat{\boldsymbol{x}}]_i > 0 \\ \text{-1} & \text{if } [\boldsymbol{\Omega}\hat{\boldsymbol{x}}]_i < 0 \\ [0, \infty) & \text{if } [\boldsymbol{\Omega}\hat{\boldsymbol{x}}]_i = 1 \\ (\text{-}\infty, 0] & \text{if } [\boldsymbol{\Omega}\hat{\boldsymbol{x}}]_i = \text{-1}. \end{cases} \tag{A.12}$$

Ref. [126] provides a more general version of the dual function for non-identity system matrices.

Above, we derived the saddle-point and dual problems using the equality of the biconjugate and the original function for a convex regularizer. The dual problem can also be derived using Lagrangian theory, as shown in [126]. Define an auxiliary (split) variable that is constrained to equal the filtered signal, *i.e.*, $\boldsymbol{z} = \boldsymbol{\Omega}\boldsymbol{x}$. Considering the specific case of the 1-norm regularizer, the Lagrangian of the constrained version of (A.7) is

$$\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 + \|\boldsymbol{z}\|_1 + \boldsymbol{d}'(\boldsymbol{\Omega}\boldsymbol{x} - \boldsymbol{z}),$$

where $\boldsymbol{d} \in \mathbb{R}^F$ is a vector of Lagrange multipliers and we have omitted the KKT conditions. Minimizing the Lagrangian with respect to $\boldsymbol{x}$ and $\boldsymbol{z}$ yields the conjugate functions for the data-fit term and 1-norm and thus the dual problem.

Using the Lagrangian perspective to derive the dual problem yields a useful relation between the filtered signal and the dual variable [126]. Because the split variable $\boldsymbol{z}$ is constrained to equal $\boldsymbol{\Omega}\boldsymbol{x}$, $[\boldsymbol{\Omega}\boldsymbol{x}]_i > 0$ implies $z_i > 0$. From (A.5), z_i is only positive and finite when $d_i = 1$. A similar argument holds for $[\boldsymbol{\Omega}\boldsymbol{x}]_i < 0$. Therefore, the dual variable and $\hat{\boldsymbol{x}}$ are related by

$$d_i \in \begin{cases} \text{sign}([\boldsymbol{\Omega}\boldsymbol{x}]_i) & \text{if } [\boldsymbol{\Omega}\hat{\boldsymbol{x}}]_i \neq 0 \\ [\text{-1}, 1] & \text{if } [\boldsymbol{\Omega}\hat{\boldsymbol{x}}]_i = 0. \end{cases} \tag{A.13}$$

The second case follows from observing that d_i can take any value in its constrained range when $z_i = 0$ as the minimum in (A.9) will be 0 regardless of d_i.

The primal-dual results reviewed in this appendix are referenced in Section 2.2.3 to relate analysis and synthesis regularizers, Section 4.3 to re-write the lower-level minimizer as a differentiable function of itself and $\boldsymbol{\gamma}$, and in Section 4.4.2 to unroll a differentiable algorithm for a non-smooth cost function.

B

Forward and Reverse Approaches to Unrolling

This appendix provides background on the forward and backward approaches to the unrolled gradient computation introduced in Section 4.4. From (4.18), the gradient of interest is:

$$\nabla \ell(\boldsymbol{\gamma}) = \nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}^{(T)}) + \left(\sum_{t=1}^{T} \left(\boldsymbol{H}_T \cdots \boldsymbol{H}_{t+1} \right) \boldsymbol{J}_t \right)' \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}^{(T)}) \in \mathbb{F}^R. \tag{B.1}$$

If one uses a gradient descent based algorithm to optimize the lower-level cost function Φ, then $\boldsymbol{H}_t = \nabla_{\boldsymbol{x}} \Psi(\boldsymbol{x}^{(t-1)}\,; \boldsymbol{\gamma}) \in \mathbb{F}^{N \times N}$ is closely related to the Hessian of Φ and $\boldsymbol{J}_t = \nabla_{\boldsymbol{\gamma}} \Psi(\boldsymbol{x}^{(t-1)}\,; \boldsymbol{\gamma}) \in \mathbb{F}^{N \times R}$ is proportional to the Jacobian of the gradient.

To compare the forward and reverse approaches to gradient computation for unrolled methods, we introduce notation for an ordered product of matrices. We indicate the arrangement of the multiplications by the set endpoints, $s \in [s_1 \leftrightarrow s_2]$ with the left endpoint, s_1, corresponding to the index for the left-most matrix in the product and the right endpoint, s_2, corresponding to the right-most matrix. Thus, for

any sequence of square matrices $\{\boldsymbol{A}\}_i$:

$$\prod_{s\in[t\leftrightarrow T]} \boldsymbol{A}_s \coloneqq \boldsymbol{A}_t\boldsymbol{A}_{t+1}\cdots\boldsymbol{A}_T = \left(\boldsymbol{A}_T'\boldsymbol{A}_{T-1}'\cdots\boldsymbol{A}_t'\right)' = \left(\prod_{s\in[T\leftrightarrow t]} \boldsymbol{A}_s'\right)'.$$

The above double arrow notation does not indicate order of operations. In the following notation the arrow direction does not affect the product result (ignoring finite precision effects), but rather signifies the direction (order) of calculation:

$$\prod_{s\in[T\leftarrow t]} \boldsymbol{A}_s \coloneqq \boldsymbol{A}_T\left(\boldsymbol{A}_{T-1}\cdots\left(\boldsymbol{A}_{t+1}\left(\boldsymbol{A}_t\right)\right)\right)$$
$$\prod_{s\in[T\rightarrow t]} \boldsymbol{A}_s \coloneqq \left(\left(\left(\boldsymbol{A}_T\boldsymbol{A}_{T-1}\right)\cdots\right)\boldsymbol{A}_{t+1}\right)\boldsymbol{A}_t.$$

We use a similar arrow notation to denote the order that terms are computed for sums; as above, the order is only important for computational considerations and does not affect the final result.

Using this notation, the reverse gradient calculation of (B.1) is

$$\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)}) + \sum_{t\in[T\rightarrow 1]} \boldsymbol{J}_t{}' \left(\prod_{s\in[(t+1)\leftarrow T]} \boldsymbol{H}_s'\right)\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)}). \tag{B.2}$$

This expression requires $\prod_{s\in[(T+1)\leftarrow T]}\boldsymbol{H}_s' = \boldsymbol{I}$, because $\boldsymbol{H}_{T+1}$ is not defined. For example, for $T=3$, we have

$$\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(3)}) + \underbrace{\boldsymbol{J}_3'(\boldsymbol{I})\boldsymbol{g}}_{t=3} + \underbrace{\boldsymbol{J}_2'\left(\boldsymbol{H}_3'\right)\boldsymbol{g}}_{t=2} + \underbrace{\boldsymbol{J}_1'\left(\boldsymbol{H}_2'\boldsymbol{H}_3'\right)\boldsymbol{g}}_{t=1},$$

where $\boldsymbol{g}$ is shorthand for $\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)})$ here. This version is called reverse as all computations (arrows) begin at the end, T.

The primary benefit of the reverse mode comes from the ability to group $\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)})$ with the right-most $\boldsymbol{H}_T$, such that all products are matrix-vector products, as seen in Fig. B.1 Further, one can save the matrix-vector products for use during the next iteration and avoid duplicating the computation. Continuing the example for $T=3$, we have

$$\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(3)}) + \underbrace{\boldsymbol{J}_3'(\boldsymbol{I})\boldsymbol{g}}_{t=1} + \underbrace{\boldsymbol{J}_2'(\overbrace{\boldsymbol{H}_3'\boldsymbol{g}}^{\Delta})}_{t=2} + \underbrace{\boldsymbol{J}_1'(\boldsymbol{H}_2'(\overbrace{\boldsymbol{H}_3'\boldsymbol{g}}^{\Delta}))}_{t=3},$$

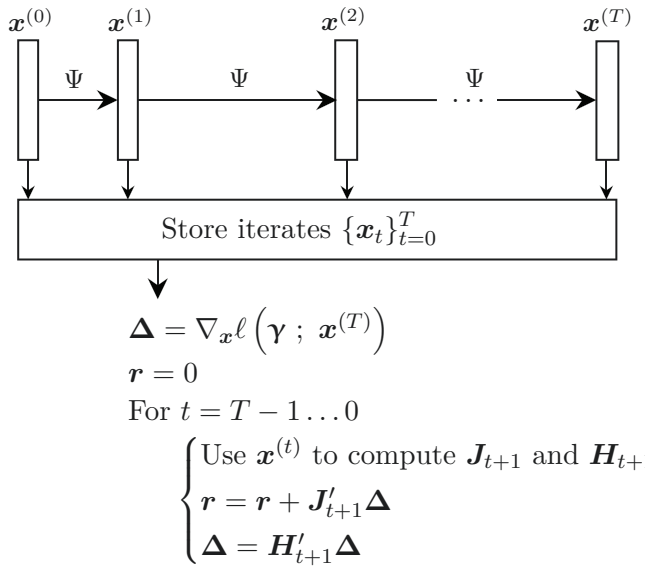


Figure B.1: Reverse mode computation of the unrolled gradient from (B.1). The first gradient computation requires $\boldsymbol{x}^{(T)}$, so all computations occur after the lower-level optimization algorithm is complete. The final gradient is $\nabla \ell(\boldsymbol{\gamma}) = \nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}^{(T)}) + \boldsymbol{r}$.

where one only needs to compute $\boldsymbol{\Delta}$ once. This ability to rearrange the parenthesis to compute matrix-vector products greatly decreases the computational requirement compared to matrix-matrix products. Excluding the costs of the optimization algorithm steps and forming the $\boldsymbol{H}_s$ and $\boldsymbol{J}_t$ matrices (these costs will be the same in the forward mode computation), reverse mode requires $\mathcal{O}(T)$ Hessian-vector multiplies and $\mathcal{O}(TNR)$ additional multiplies. The trade-off is that reverse mode requires storing all T iterates, $\boldsymbol{x}^{(t)}$, so that one can compute the corresponding Hessians and Jacobians from them as needed, and thus has a memory complexity $\mathcal{O}(TN)$.

The forward mode calculation of (B.1), depicted in Fig. B.2, has all computations (arrows) starting at the earlier iterate:

$$\nabla_{\boldsymbol{\gamma}} \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}^{(T)}) + \left(\sum_{t \in [1 \to T]} \left(\prod_{s \in [T \leftarrow (t+1)]} \boldsymbol{H}_s \right) \boldsymbol{J}_t \right)' \nabla_{\boldsymbol{x}} \ell(\boldsymbol{\gamma}\,; \boldsymbol{x}^{(T)}). \quad \text{(B.3)}$$

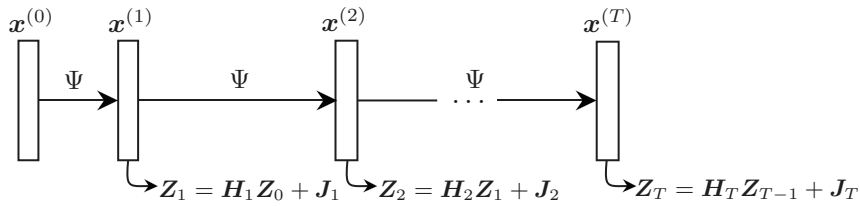


Figure B.2: Forward mode computation of the unrolled gradient from (B.3). The intermediate computation matrix, $\boldsymbol{Z}$, is initialized to zero ($\boldsymbol{Z}_0 = \boldsymbol{0}$) then updated every iteration. The final gradient is $\nabla\ell(\boldsymbol{\gamma}) = \nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)}) + \boldsymbol{Z}_T'\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)})$.

As before, $\boldsymbol{H}_{T+1}$ is not defined, so we take $\prod_{s\in[T\leftarrow(T+1)]}\boldsymbol{H}_s = \boldsymbol{I}$. For example, for $T = 3$ we have

$$\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)}) + \left(\underbrace{((\boldsymbol{H}_3\boldsymbol{H}_2)\boldsymbol{J}_1)'}_{t=1} + \underbrace{((\boldsymbol{H}_3)\boldsymbol{J}_2)'}_{t=2} + \underbrace{((\boldsymbol{I})\boldsymbol{J}_3)'}_{t=3}\right)\boldsymbol{g}.$$

How the forward mode avoids storing $\boldsymbol{x}$ iterates is evident after rearranging the parenthesis to avoid duplicate calculations, as illustrated in Fig. B.2. Continuing the example for $T = 3$, we have

$$\nabla_{\boldsymbol{\gamma}}\ell(\boldsymbol{\gamma}\,;\boldsymbol{x}^{(T)}) + \left[\underbrace{\boldsymbol{H}_3\left(\overbrace{\boldsymbol{H}_2\underbrace{(\boldsymbol{H}_1\cdot\boldsymbol{0}+\boldsymbol{J}_1)}_{\boldsymbol{Z}_1}+\boldsymbol{J}_2}^{\boldsymbol{Z}_2}\right)+\boldsymbol{J}_3}_{\boldsymbol{Z}_3}\right]'\boldsymbol{g},$$

where $\boldsymbol{Z}_s = \boldsymbol{H}_s\boldsymbol{Z}_{s-1} + \boldsymbol{J}_s \in \mathbb{F}^{N\times R}$ stores the intermediate calculations. The above formula also illustrates why $\boldsymbol{H}_1$ is not needed in (4.17); $\nabla_{\boldsymbol{\gamma}}\boldsymbol{x}^{(0)} = \boldsymbol{0}$ is the last element from applying the chain rule.

There is no way to rearrange the terms in the forward mode formula to achieve matrix-vector products (while preserving the computation order). Therefore, the computation requirement is much higher at $\mathcal{O}(TR)$ Hessian-vector multiplications. The corresponding benefit of the forward mode method is that it does not require storing iterates, thus decreasing (in the common case when $T > R$) the memory requirement to $\mathcal{O}(NR)$ for storing the intermediate matrix $\boldsymbol{Z}_s$ during calculation.

As with the minimizer approach in Section 4.2, the computational complexity of the unrolled approach is lower than the generic bound when we consider the specific example of learning convolutional filters according to (Ex). Nevertheless, the general comparison that reverse mode takes more memory but less computation holds true. See Tab. 4.1 for a comparison of the computational and memory complexities.

C

Additional Running Example Results

This appendix derives some results that are relevant to the running example used throughout the survey.

C.1 Derivatives for Convolutional Filters

This section proves the result

$$\frac{\partial}{\partial c_{\boldsymbol{s}}} \left(\tilde{\boldsymbol{c}}_k \circledast f.(\boldsymbol{c}_k \circledast \boldsymbol{x})\right) = f.(\boldsymbol{c}_k \circledast \boldsymbol{z}^{\langle \boldsymbol{s} \rangle}) + \tilde{\boldsymbol{c}}_k \circledast \left(\dot{f}.(\boldsymbol{c}_k \circledast \boldsymbol{x}) \odot \boldsymbol{x}^{\langle -\boldsymbol{s} \rangle}\right), \tag{C.1}$$

when considering $\mathbb{F} = \mathbb{R}$. This equation is key to finding derivatives of the lower-level cost function in (Ex) with respect to the filter coefficients.

To simplify notation, we drop the indexing over k, so $\boldsymbol{c}$ is a single filter and $c_{\boldsymbol{s}}$ denotes the $\boldsymbol{s}$th element in the filter for $\boldsymbol{s} \in \mathbb{Z}^D$. Here, $\boldsymbol{s}$ indexes every dimension of $\boldsymbol{c}$, *e.g.*, for a two-dimensional filter, we could equivalently write $\boldsymbol{s}$ as $\langle s_1, s_2 \rangle$. Recall that the notation $\tilde{\boldsymbol{c}}$ signifies a reversed version of $\boldsymbol{c}$, as needed for the adjoint of convolution.

Define the notation $\boldsymbol{x}^{\langle \boldsymbol{i} \rangle}$ as the vector $\boldsymbol{x}$ circularly shifted according to the index $\boldsymbol{i}$. Thus, if $\boldsymbol{x}$ is 0-indexed and we use circular indexing,

$$(\boldsymbol{x}^{\langle \boldsymbol{s} \rangle})_{\boldsymbol{i}} = \boldsymbol{x}_{\boldsymbol{i}-\boldsymbol{s}}.$$

As two examples,

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} \rightarrow \boldsymbol{x}^{\langle \text{-}1 \rangle} = \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_N \\ x_1 \end{bmatrix},$$

and, in two dimensions, if $\boldsymbol{X} \in \mathbb{F}^{M \times N}$

$$\boldsymbol{X}^{\langle 1,2 \rangle} = \begin{bmatrix} x_{M,N-1} & x_{M,N} & x_{M,1} & \cdots & x_{M,3} \\ x_{1,N-1} & x_{1,N} & x_{1,1} & \cdots & x_{1,3} \\ x_{2,N-1} & x_{2,N} & x_{2,1} & \cdots & x_{2,3} \\ \vdots & & \ddots & & \vdots \\ x_{M-1,N-1} & x_{M-1,N} & x_{M-1,1} & \cdots & x_{M-1,3} \end{bmatrix}.$$

This circular shift notation is useful in the derivation and statement of the desired gradient.

Define $\boldsymbol{z} = \boldsymbol{c} \circledast \boldsymbol{x}$, where $\boldsymbol{c}$ and $\boldsymbol{x}$ are both N-dimensional. By the definition of convolution, $\boldsymbol{z}$ is given by

$$\boldsymbol{z} = \sum_{i_1} \cdots \sum_{i_N} c_{i_1,\ldots,i_N} \boldsymbol{x}^{\langle \text{-}i_1,\ldots,\text{-}i_N \rangle} := \sum_{i_1,\ldots,i_N} c_{i_1,\ldots,i_N} \boldsymbol{x}^{\langle \text{-}\boldsymbol{i} \rangle},$$

where, for each sum, the indexing variable i_n iterates over the size of $\boldsymbol{c}$ in the ith dimension and we simplify the index for circularly shifting vectors, $i_1, \ldots, i_N$, as simply $\langle \boldsymbol{i} \rangle$. This expression shows that the derivative of $\boldsymbol{c} \circledast \boldsymbol{x}$ with respect to the $\boldsymbol{s}$th filter coefficient is the $\text{-}\boldsymbol{s}$th coefficient in $\boldsymbol{x}$, *i.e.*,

$$\frac{\partial}{\partial c_{\boldsymbol{s}}} (\boldsymbol{c} \circledast \boldsymbol{x}) = \boldsymbol{x}^{\langle -\boldsymbol{s} \rangle}. \tag{C.2}$$

We can now find the partial derivative of interest:

$$\begin{aligned}
\tilde{\boldsymbol{c}} \circledast f.(\boldsymbol{z}) &= \sum_{i_1,\ldots,i_N} [\tilde{\boldsymbol{c}}]_{i_1,\ldots,i_N} f.(\boldsymbol{z})^{\langle \text{-}\boldsymbol{i}\rangle} && \text{by the convolution formula} \\
&= \sum_{i_1,\ldots,i_N} [\tilde{\boldsymbol{c}}]_{i_1,\ldots,i_N} f.\left(\boldsymbol{z}^{\langle \text{-}\boldsymbol{i}\rangle}\right) && \text{since } f \text{ operates point-wise} \\
&= \sum_{i_1,\ldots,i_N} c_{\text{-}i_1,\ldots,\text{-}i_N} f.\left(\boldsymbol{z}^{\langle \text{-}\boldsymbol{i}\rangle}\right) && \text{by definition of } \tilde{\boldsymbol{c}} \\
&= \sum_{i_1,\ldots,i_N} c_{i_1,\ldots,i_N} f.\left(\boldsymbol{z}^{\langle \boldsymbol{i}\rangle}\right) && \text{reverse summation order.}
\end{aligned}$$

Recall that $\boldsymbol{z}$ is a function of $c_{\boldsymbol{s}}$. Therefore, using the chain rule to take the derivative,

$$\begin{aligned}
&\frac{\partial}{\partial c_{\boldsymbol{s}}} \left(\tilde{\boldsymbol{c}} \circledast f.(\boldsymbol{z})\right) \\
&\quad = f.(\boldsymbol{z}^{\langle \boldsymbol{s}\rangle}) + \sum_{i_1} \cdots \sum_{i_N} c_{i_1,\ldots,i_N} \dot{f}.(\boldsymbol{z}^{\langle i_1,\ldots,i_N\rangle}) \odot \nabla_{c_{\boldsymbol{s}}} \left(\boldsymbol{z}^{\langle \boldsymbol{i}\rangle}\right) \\
&\quad = f.(\boldsymbol{z}^{\langle \boldsymbol{s}\rangle}) + \sum_{i_1} \cdots \sum_{i_N} [\tilde{\boldsymbol{c}}]_{\text{-}i_1,\ldots,\text{-}i_N} \dot{f}.(\boldsymbol{z}^{\langle i_1,\ldots,i_N\rangle}) \odot \boldsymbol{x}^{\langle \boldsymbol{i}-\boldsymbol{s}\rangle},
\end{aligned}$$

where the second equality follows from (C.2) and the definition of $\tilde{\boldsymbol{c}}$. Recognizing the convolution formula in the second summand, the expression can be simplified to

$$f.(\boldsymbol{z}^{\langle \boldsymbol{s}\rangle}) + \tilde{\boldsymbol{c}} \circledast \left(\dot{f}.(\boldsymbol{z}) \odot \boldsymbol{x}^{\langle -\boldsymbol{s}\rangle}\right).$$

This proves the claim. Note that the provided formula is for a single element in $\boldsymbol{c}$. One can concatenate the partial derivative result for each value of $\boldsymbol{s}$ to get the full Jacobian.

C.2 Evaluating Assumptions for the Running Example

To better understand the upper-level assumptions Aℓ1-Aℓ3 and lower-level assumptions AΦ1-AΦ6 in Section 5.3.1, this section examines whether the filter learning example (Ex) meets each assumption.

C.2.1 Upper-level Loss Assumptions

Recall the upper-level loss function in (Ex) is squared error:

$$\ell(\boldsymbol{\gamma}\,;\,\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}^{\text{true}}\|_2^2, \tag{C.3}$$

where ℓ is typically evaluated at $\boldsymbol{x} = \hat{\boldsymbol{x}}(\boldsymbol{\gamma})$.

The loss function (C.3) satisfies Aℓ1. Because there is no dependence on $\boldsymbol{\gamma}$ in the upper-level, $L_{\boldsymbol{x},\nabla_{\boldsymbol{\gamma}}\ell} = 0$. The gradient with respect to $\boldsymbol{x}$ is $\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\,\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{x}^{\text{true}}$, so $L_{\boldsymbol{x},\nabla_{\boldsymbol{x}}\ell} = 1$.

The norm of the upper-level gradient with respect to $\boldsymbol{x}$,

$$\|\nabla_{\boldsymbol{x}}\ell(\boldsymbol{\gamma}\,;\,\boldsymbol{x})\| = \left\|\boldsymbol{x} - \boldsymbol{x}^{\text{true}}\right\|,$$

can grow arbitrarily large, so condition Aℓ2 is not met in general. However, in most applications, one can assume an upper bound (possibly quite large) on the elements of $\boldsymbol{x}^{\text{true}}$ and impose that bound as a box constraint when computing $\hat{\boldsymbol{x}}$. Then the triangle inequality provides a bound on $\|\boldsymbol{x} - \boldsymbol{x}^{\text{true}}\|$ for all $\boldsymbol{x}$ within the constraint box.

Finally, Aℓ3 is met by any loss function, including (C.3), that lacks cross terms between $\boldsymbol{x}$ and $\boldsymbol{\gamma}$. We are unaware of any bilevel method papers using such cross terms.

C.2.2 Lower-level Cost Assumptions

One property used below in many of the bounds for the lower-level cost function is that

$$\sigma_1(\boldsymbol{C}_k) = \|\boldsymbol{c}_k\|_1\,, \tag{C.4}$$

where $\sigma_1(\cdot)$ is a function that returns the first singular value of its matrix argument. This property follows from Young's inequality and is related to bounded-input bounded-output stability of linear and time invariant systems [203].

As with the upper-level assumptions considered above, (Ex) meets the lower-level assumptions AΦ1-AΦ6 if we impose additional constraints on the maximum norm of variables. In addition to bounding the elements in $\boldsymbol{x}$, as we did to ensure Aℓ2, imposing bounds on $\|\boldsymbol{c}_k\|$ and $|\beta_k|$ is sufficient to meet all the lower-level assumptions. We now examine each condition individually.

Recall from (Ex) that the example lower-level cost function is

$$\hat{\boldsymbol{x}}(\boldsymbol{\gamma}) = \underset{\boldsymbol{x}\in\mathbb{F}^N}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\|_2^2 + e^{\beta_0}\sum_{k=1}^{K} e^{\beta_k}\mathbf{1}'\phi.(\boldsymbol{c}_k \circledast \boldsymbol{x};\epsilon),$$

where ϕ is a corner-rounded 1-norm (CR1N).

As described in Section 4.2, the minimizer approach requires Φ to be twice differentiable. Thus, Φ satisfies AΦ1. This condition limits the choices of ϕ to twice differentiable functions.

Considering AΦ2, the gradient of Φ with respect to $\boldsymbol{x}$ is Lipschitz continuous in $\boldsymbol{x}$ if the norm of the Hessian, $\|\nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi(\boldsymbol{x};\boldsymbol{\gamma})\|_2$, is bounded. Using (4.9) and assuming the Lipschitz constant of the derivative of ϕ is $L_{\dot{\phi}}$ (for (CR1N), $L_{\dot{\phi}} = \frac{1}{\epsilon}$), a Lipschitz constant for $\nabla_{\boldsymbol{x}}\Phi$ is

$$\begin{aligned} L_{\boldsymbol{x},\nabla_{\boldsymbol{x}}\Phi} &= \sigma_1^2(\boldsymbol{A}) + L_{\dot{\phi}}e^{\beta_0}\sum_k e^{\beta_k}\sigma_1(\boldsymbol{C}_k'\boldsymbol{C}_k) \\ &= \sigma_1^2(\boldsymbol{A}) + L_{\dot{\phi}}e^{\beta_0}\sum_k e^{\beta_k}\|\boldsymbol{c}_k\|_1^2 \text{ by (C.4).} \end{aligned} \tag{C.5}$$

The Lipschitz constant $L_{\boldsymbol{x},\nabla_{\boldsymbol{x}}\Phi}$ depends on the values in $\boldsymbol{\gamma}$ and therefore does not strictly satisfy AΦ2. Here if β_0, β_k, and $\boldsymbol{c}_k$ have upper bounds, then one can upper bound $L_{\boldsymbol{x},\nabla_{\boldsymbol{x}}\Phi}$. All of the bounds below have similar considerations.

To consider the strong convexity condition in AΦ3, we consider the Hessian,

$$\nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi(\boldsymbol{x};\boldsymbol{\gamma}) = \underbrace{\boldsymbol{A}'\boldsymbol{A}}_{\text{From data-fit term}} + \underbrace{e^{\beta_0}\sum_k e^{\beta_k}\boldsymbol{C}_k'\operatorname{diag}(\ddot{\phi}.(\boldsymbol{c}_k \circledast \boldsymbol{x}))\boldsymbol{C}_k}_{\text{From regularizer}}. \tag{C.6}$$

We assume that $\ddot{\phi}(z) \geq 0\,\forall z$, as is the case for the corner rounded 1-norm. If $\boldsymbol{A}'\boldsymbol{A}$ is positive-definite with $\sigma_N(\boldsymbol{A}'\boldsymbol{A}) > 0$ (this is equivalent to $\boldsymbol{A}$ having full column rank), then the Hessian is positive-definite and $\mu_{\boldsymbol{x},\Phi} = \sigma_N^2(\boldsymbol{A})$ suffices as a strong convexity parameter. In applications like compressed sensing, $\boldsymbol{A}$ does not have full column rank. In such cases, $\sigma_N(\boldsymbol{A}'\boldsymbol{A}) = 0$ and as $e^{\beta_0} \to 0$ the regularizer term vanishes, so there does not exist any universal $\mu_{\boldsymbol{x},\Phi} > 0$ for all $\boldsymbol{\gamma} \in \mathbb{F}^R$, so the strong convexity condition AΦ3 is not satisfied. However, as discussed

in Section 4.2.3, the condition may hold in practice for many values of $\boldsymbol{\gamma}$. How to adapt the complexity theory to rigorously address these subtleties is an open question.

The fourth condition, AΦ4, is that $\nabla_{\boldsymbol{xx}}\Phi(\boldsymbol{x};\boldsymbol{\gamma})$ and $\nabla_{\boldsymbol{\gamma x}}\Phi(\boldsymbol{x};\boldsymbol{\gamma})$ are Lipschitz continuous with respect to $\boldsymbol{x}$ for all $\boldsymbol{\gamma}$. For the first part part, a Lipschitz constant results from bounding the difference in the Hessian evaluated at two points, $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$:

$$\begin{aligned}&\left\|\nabla_{\boldsymbol{xx}}\Phi(\boldsymbol{x}^{(1)};\boldsymbol{\gamma}) - \nabla_{\boldsymbol{xx}}\Phi(\boldsymbol{x}^{(2)};\boldsymbol{\gamma})\right\|_2 \\ &\quad = \left\| e^{\beta_0}\sum_k e^{\beta_k}\boldsymbol{C}_k'\mathrm{diag}(\ddot{\phi}.(\boldsymbol{c}_k \circledast \boldsymbol{x}^{(1)}) - \ddot{\phi}(\boldsymbol{c}_k \circledast \boldsymbol{x}^{(2)}))\boldsymbol{C}_k \right\|_2 .\end{aligned}$$

Since every element of $\ddot{\phi}$ is bounded in $(0, L_{\dot{\phi}})$, the difference between any two evaluations of $\ddot{\phi}$ is at most $L_{\dot{\phi}}$. Thus

$$\begin{aligned}\left\|\nabla_{\boldsymbol{xx}}\Phi(\boldsymbol{x}^{(1)};\boldsymbol{\gamma}) - \nabla_{\boldsymbol{xx}}\Phi(\boldsymbol{x}^{(2)};\boldsymbol{\gamma})\right\|_2 &\leq e^{\beta_0}L_{\dot{\phi}}\sum_k e^{\beta_k}\left\|\boldsymbol{C}_k'\boldsymbol{C}_k\right\|_2 \\ &\leq e^{\beta_0}L_{\dot{\phi}}\sum_k e^{\beta_k}\left\|\boldsymbol{c}_k\right\|_1^2 .\end{aligned}$$

The final simplification again uses (C.4). Thus,

$$L_{\boldsymbol{x},\nabla_{\boldsymbol{xx}}\Phi} = e^{\beta_0}L_{\dot{\phi}}\sum_k e^{\beta_k}\left\|\boldsymbol{c}_k\right\|_1^2 .$$

For the second part of AΦ4, we must look at the tuning parameters and filter coefficients separately. When considering learning a tuning parameter, β_k,

$$\nabla_{\beta_k\boldsymbol{x}}\Phi(\boldsymbol{x};\boldsymbol{\gamma}) = e^{\beta_0+\beta_k}\boldsymbol{C}_k'\dot{\phi}.(\boldsymbol{C}_k\boldsymbol{x}).$$

To find a Lipschitz constant, consider the Jacobian:

$$\nabla_{\boldsymbol{x}}\left(\nabla_{\beta_k\boldsymbol{x}}\Phi(\boldsymbol{x};\boldsymbol{\gamma})\right) = e^{\beta_0+\beta_k}\boldsymbol{C}_k'\mathrm{diag}(\ddot{\phi}.(\boldsymbol{C}_k\boldsymbol{x}))\boldsymbol{C}_k.$$

A Lipschitz constant of $\nabla_{\beta_k\boldsymbol{x}}\Phi(\boldsymbol{x};\boldsymbol{\gamma})$ is given by the bound on the norm of this matrix (we chose to use the matrix 2-norm, also called the spectral norm). Using similar steps as above to simplify the expression, $L_{\boldsymbol{x},\nabla_{\beta_k\boldsymbol{x}}\Phi} = e^{\beta_0+\beta_k}L_{\dot{\phi}}\left\|\boldsymbol{c}_k\right\|_1^2$.

When considering learning the $\boldsymbol{s}$th element of the kth filter,

$$\nabla_{c_{k,\boldsymbol{s}}\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma}) = e^{\beta_0+\beta_k}\left(\dot{\phi}.((\boldsymbol{C}_k\boldsymbol{x})^{\langle s\rangle}) + \boldsymbol{C}_k'\left(\ddot{\phi}.(\boldsymbol{C}_k\boldsymbol{x})\odot\boldsymbol{x}^{\langle \text{-}s\rangle}\right)\right)$$

$$= e^{\beta_0+\beta_k}\left(\underbrace{\dot{\phi}.(\boldsymbol{R}_1\boldsymbol{C}_k\boldsymbol{x})}_{\text{Expression 1}} + \underbrace{\boldsymbol{C}_k'\left(\ddot{\phi}.(\boldsymbol{C}_k\boldsymbol{x})\odot\boldsymbol{R}_2\boldsymbol{x}\right)}_{\text{Expressions 2-3}}\right) \in \mathbb{F}^N,$$

where $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are rotation matrices that depends on $\boldsymbol{s}$ such that $\boldsymbol{R}_1\boldsymbol{x} = \boldsymbol{x}^{\langle \boldsymbol{s}\rangle}$ and $\boldsymbol{R}_2\boldsymbol{x} = \boldsymbol{x}^{\langle \text{-}\boldsymbol{s}\rangle}$. For taking the gradient, it is convenient to note that the last term can be expressed in multiple ways:

$$\ddot{\phi}.(\boldsymbol{C}_k\boldsymbol{x})\odot\boldsymbol{x}^{\langle \text{-}\boldsymbol{s}\rangle} = \underbrace{\text{diag}(\ddot{\phi}.(\boldsymbol{C}_k\boldsymbol{x}))\boldsymbol{R}_2\boldsymbol{x}}_{\text{Expression 2}} = \underbrace{\text{diag}(\boldsymbol{R}_2\boldsymbol{x})\ddot{\phi}.(\boldsymbol{C}_k\boldsymbol{x})}_{\text{Expression 3}}.$$

Using the alternate expressions to perform the chain rule with respect to the $\boldsymbol{x}$ term that is not in the diag$(\cdot)$ statement, the gradient with respect to $\boldsymbol{x}$ is:

$$\nabla_{\boldsymbol{x}}\left(\nabla_{c_{k,\boldsymbol{s}}\boldsymbol{x}}\Phi(\boldsymbol{x}\,;\boldsymbol{\gamma})\right) = e^{\beta_0+\beta_k}\Big(\underbrace{\boldsymbol{C}_k'\boldsymbol{R}_1'\text{diag}(\dddot{\phi}.(\boldsymbol{R}_1\boldsymbol{C}_k\boldsymbol{x}))}_{\text{Expression 1}}$$
$$+\underbrace{\boldsymbol{C}_k'\text{diag}(\ddot{\phi}.(\boldsymbol{C}_k\boldsymbol{x}))\boldsymbol{R}_2}_{\text{Expression 2}}$$
$$+\underbrace{\boldsymbol{C}_k'\text{diag}(\dddot{\phi}(\boldsymbol{C}_k\boldsymbol{x}))\text{diag}(\boldsymbol{R}_2\boldsymbol{x})'\boldsymbol{C}_k}_{\text{Expression 3}}\Big).$$

The bound on the spectral norm of the first and second expressions are both $\sigma_1(\boldsymbol{C}_k)L_{\dot{\phi}}$ because, for any $\boldsymbol{z}\in\mathbb{F}^N$,

$$\|\text{diag}(\ddot{\phi}.(\boldsymbol{z}))\|_2 \leq \max_z|\ddot{\phi}(z)| = L_{\dot{\phi}}.$$

The third expression is bounded by $\sigma_1^2(\boldsymbol{C}_k)\,\|\boldsymbol{x}\|_2\,L_{\ddot{\phi}}$, which requires a bound on the norm of $\boldsymbol{x}$, similar to Aℓ2. Summing the three expressions and including the tuning parameters gives the final Lipschitz constant

$$L_{\boldsymbol{x},\nabla_{c_{k,\boldsymbol{s}}\boldsymbol{x}}\Phi} = e^{\beta_0+\beta_k}\sigma_1(\boldsymbol{C}_k)(2L_{\dot{\phi}} + \sigma_1(\boldsymbol{C}_k)L_{\ddot{\phi}}\,\|\boldsymbol{x}\|_2). \tag{C.7}$$

The fifth assumption, AΦ5 states that the mixed second gradient of Φ is bounded. For the tuning parameters, the mixed second gradient is given in (4.9) as

$$\nabla_{\beta_k\boldsymbol{x}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}) = e^{\beta_0}e^{\beta_k}\tilde{\boldsymbol{c}}_k \circledast \dot{\phi}.(\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}}).$$

The bound given in AΦ5 follows easily by considering that

$$\|\mathrm{diag}(\dot{\phi}.(\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}}))\|_2 \leq \max_z |\dot{\phi}(z)| = L_\phi.$$

For a filter coefficient, the mixed second gradient is more complicated:

$$\nabla_{c_{k,\boldsymbol{s}}\boldsymbol{x}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma}) = e^{\beta_0+\beta_k}\Big(\underbrace{\dot{\phi}.((\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}})^{\langle \boldsymbol{s}\rangle})}_{\text{Bounded by } L_\phi} + \tilde{\boldsymbol{c}}_k \circledast \Big(\underbrace{\ddot{\phi}.(\boldsymbol{c}_k \circledast \hat{\boldsymbol{x}})}_{\text{Bounded by } L_{\dot{\phi}}} \odot \hat{\boldsymbol{x}}^{\langle -\boldsymbol{s}\rangle}\Big)\Big).$$

Assuming that the bounds L_ϕ and $L_{\dot{\phi}}$ exist (they are 1 and $\frac{1}{\epsilon}$ respectively for (CR1N)), a bound on the norm of the mixed gradient is

$$\|\nabla_{c_{k,\boldsymbol{s}}\boldsymbol{x}}\Phi(\hat{\boldsymbol{x}}\,;\boldsymbol{\gamma})\|_2 \leq e^{\beta_0+\beta_k}\left(L_\phi + L_{\dot{\phi}}\,\|\boldsymbol{c}_k\|_1\,\|\boldsymbol{x}\|_2\right).$$

The sixth assumption, AΦ6, is that $L_{\boldsymbol{\gamma},\nabla_{\boldsymbol{\gamma}\boldsymbol{x}}\Phi}$ and $L_{\boldsymbol{\gamma},\nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi}$ exist. Lipschitz constants for the tuning parameters are

$$L_{\beta_k,\nabla_{\beta_k\boldsymbol{x}}\Phi} = e^{\beta_0+\beta_k}\,\|\boldsymbol{c}_k\|_1\,L_\phi \text{ and } L_{\beta_k,\nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi} = e^{\beta_0+\beta_k}\,\|\boldsymbol{c}_k\|_1^2\,L_{\dot{\phi}}.$$

Using similar derivations as shown above, corresponding Lipschitz constants for the filter coefficients are

$$\begin{aligned} L_{c_{k,s},\nabla_{c_{k,s}\boldsymbol{x}}\Phi} &= e^{\beta_0+\beta_k}\left(L_\phi + \|\boldsymbol{x}\|_2\left(L_{\dot{\phi}} + L_{\ddot{\phi}}\,\|\boldsymbol{c}_k\|_1\,\|\boldsymbol{x}\|_2\right)\right) \\ L_{c_{k,s},\nabla_{\boldsymbol{x}\boldsymbol{x}}\Phi} &= e^{\beta_0+\beta_k}\left(2L_{\dot{\phi}}\,\|\boldsymbol{c}_k\|_1 + L_{\ddot{\phi}}\,\|\boldsymbol{c}_k\|_1^2\,\|\boldsymbol{x}\|_2\right). \end{aligned}$$

This is the last lower-level condition in Section 5.3.1 for the single-loop and double-loop bilevel optimization method analysis.

D

Implementation Details

This appendix describes the experimental settings used throughout this review. We first present the common settings; the following sub-sections detail any differences specifically for the results in Fig. 1.3 and for the series of figures using the cameraman image (Fig. 5.2, Fig. 6.1, and Fig. 6.2). The code for all experiments is available on github [204].

The experiments consider the denoising problem ($\boldsymbol{A} = \boldsymbol{I}$) and use (CR1N) as the sparsifying function ϕ with $\epsilon = 0.01$. The training data is typically on the scale [0, 1] and noisy samples are generated from the clean training data using (1.4) with zero-mean Gaussian noise with a standard deviation of $\sigma = 25/255$, following [30].

The lower-level optimizer is the optimized gradient method (OGM) with gradient-based restart [148]. We calculate the step-size based on the Lipschitz constant of the lower-level gradient using (C.5) every upper-level iteration. Each experiment sets a maximum number of lower-level iterations, but the lower-level optimization will terminate early if it converges, defined as if $\|\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}\,; \boldsymbol{\gamma})\| < 10^{-5}$.

The upper-level optimizer follows the general structure of the double-loop procedure outlined in Alg. 3. To compute $\nabla \ell(\boldsymbol{\gamma})$, we use the minimizer formulation (4.8), with the conjugate gradient (CG) method

to compute the Hessian-inverse-vector product (4.10). As suggested in [117], the initialization for the lower-level optimization is the estimated minimizer from the previous outer loop iteration, $\boldsymbol{x}^{(T)}(\boldsymbol{\gamma}^{(u\text{-}1)})$ and the initialization for the CG method is the solution from the previous CG iteration. Following [34] and other bilevel works, the experiments use Adam with the default parameters [145] to determine the size of the upper-level gradient descent; this choice avoids introducing the tuning parameter α_ℓ.

The learnable parameters include the filter coefficients and the tuning parameters β_k for $k \in [1, K]$. The experiments either use random or DCT filters to initialize $\boldsymbol{h}$. An initial grid search determines the tuning parameter β_0; β_k for $k \in [1, K]$ are initialized as 0 such that $e^{\beta_k} = 1$.

D.1 Vertical Bar Training Image

This section describes additional details for Fig. 1.3. This simple proof of concept used 50 lower-level iterations ($T = 50$) and 4,000 upper-level iterations ($U = 4,000$). The initial grid search for β_0 yielded -4.6.

When $\phi(z) = |z|$, one can absorb the kth filter's magnitude into the tuning parameter β_k because $\|\boldsymbol{c}_k \circledast \boldsymbol{x}\|_1 = \|\boldsymbol{c}_k\|_2 \left\| \frac{1}{\|\boldsymbol{c}_k\|_2} \boldsymbol{c}_k \circledast \boldsymbol{x} \right\|_1$. When using (CR1N), this equality no longer holds, but

$$e^{\beta_0 + \beta_k} \|\boldsymbol{c}_k\|_2 \tag{D.1}$$

still provides a reasonable approximation for the overall regularization strength for the kth filter. From left to right, the approximate regularization strengths of the filters in Fig. 1.3 are 0.77, 0.49, 0.17, and 0.05.

The learned filters reflect that the training data is constant along the columns. Visually, the filters resemble vertical (extended) finite differences. This matches our expectations as a filter that takes vertical finite differences will exactly sparsify the noiseless signal. Further, the maximum sum of the columns of the learned filters is 10^{-5}. In contrast, the sum of the rows of the learned filters varies from -2.6 to 3.0.

D.2 Cameraman Training Image

This section describes the experimental settings for Fig. 5.2, Fig. 6.2, and Fig. 6.1.

To reduce computation, we selected three 50×50 patches from the "cameraman" image in Fig. 6.2 to use as the training data. We hand selected the training patches to contain structure. Fig. D.1 shows the training image patches.

We set the lower-level initialization $\hat{\boldsymbol{x}}(\boldsymbol{\gamma}^{(0)})$ by optimizing the lower-level cost function until the norm of the gradient fell below a threshold for each training patch, *i.e.*, until $\frac{1}{\sqrt{N}} \left\| \nabla_{\boldsymbol{x}} \Phi \left(\hat{\boldsymbol{x}}_j(\boldsymbol{\gamma}^{(0)}) \, ; \, \boldsymbol{\gamma}^{(0)} \right) \right\|_2 < 10^{-7}$ for $j \in [1, J]$. The lower-level optimizer consisted of 10 iterations of OGM [148].

As shown in Fig. 6.1, the initial filters are the 48 non-constant DCT filters of size 7×7. The initial grid search for β_0 yielded -4. In summary, the settings are $J = 3$, $N = 50 \cdot 50$, $S = 7 \cdot 7$, $K = 48$, $R = 48(49 + 1) = 2400$, $\beta_0 = \text{-}4$, $T = 10$, and $U = 10,000$.

Fig. 6.1 shows the learned filters. To visualize the filters when $\boldsymbol{\gamma}$ includes $\boldsymbol{h}$, Fig. 6.1c scales each learned filter $\hat{\boldsymbol{c}}_k$ to have unit norm. Fig. D.2 shows the learned filters with the effective regularization strength printed above each filter.

Figure D.1: Patches from the cameraman test images used as the training dataset.

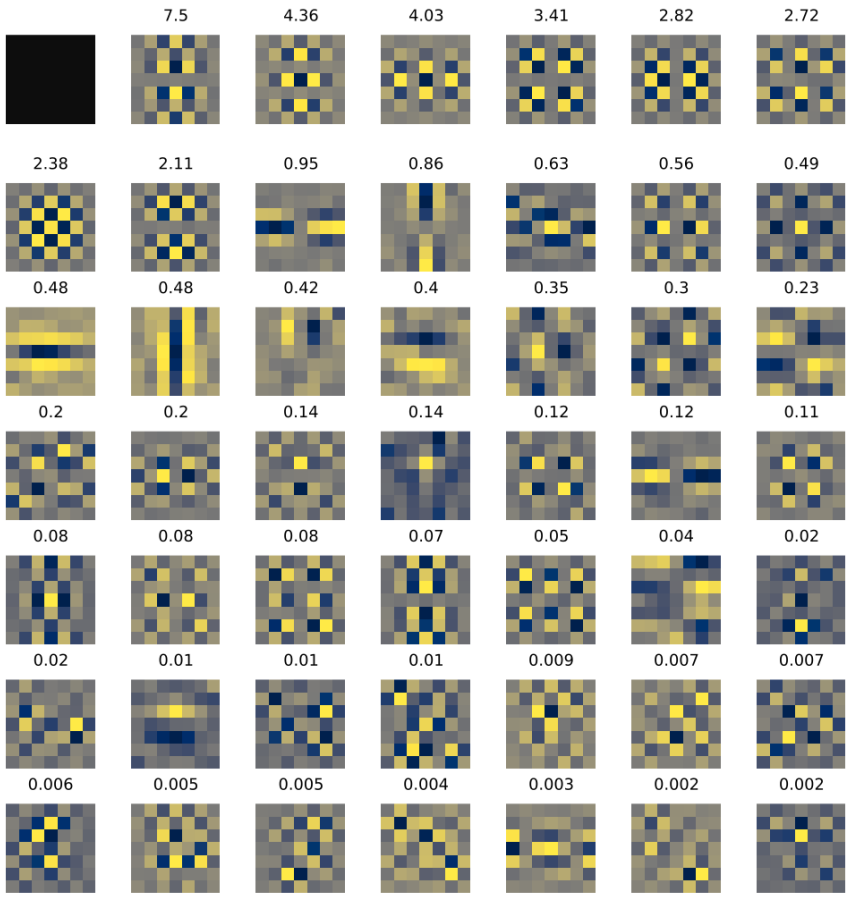


Figure D.2: Learned filers for (Ex) when $\boldsymbol{\gamma}$ includes $\boldsymbol{h}$ and $\boldsymbol{\beta}$, ordered by their effective regularization strength $e^{\beta_k} \left\| \boldsymbol{c}_k \right\|_2$, which is printed above each filter. This effective regularization does not include the influence of e^{β_0}, which is uniform across all filters.

References

[1] H. W. Engl, M. Hanke, and A. Neubauer, *Regularization of inverse problems*. Dordrecht: Kluwer, 1996.

[2] C. H. McCollough, A. C. Bartley, R. E. Carter, B. Chen, T. A. Drees, P. Edwards, D. R. Holmes, A. E. Huang, F. Khan, S. Leng, K. L. McMillan, G. J. Michalak, K. M. Nunez, L. Yu, and J. G. Fletcher, "Low-dose CT for the detection and classification of metastatic liver lesions: Results of the 2016 Low Dose CT Grand Challenge," e339–52, *Med. Phys.*, vol. 44, no. 10, Oct. 2017. DOI: 10.1002/mp.12345.

[3] Y. Eldar and G. Kutyniok, *Compressed sensing: Theory and applications*. Cambridge, 2012. DOI: 10.1017/CBO9780511794308.

[4] G. Wang, "A perspective on deep imaging," 8914–24, *IEEE Access*, vol. 4, Nov. 2016. DOI: 10.1109/ACCESS.2016.2624938.

[5] K. Hammernik and F. Knoll, "Machine learning for image reconstruction," in *Handbook of Medical Image Computing and Computer Assisted Intervention*, Elsevier, 2020, pp. 25–64. DOI: 10.1016/B978-0-12-816176-0.00007-7.

[6] S. Ravishankar, J. C. Ye, and J. A. Fessler, "Image reconstruction: From sparsity to data-adaptive methods and machine learning," 86–109, *Proc. IEEE*, vol. 108, no. 1, Jan. 2020. DOI: 10.1109/JPROC.2019.2936204.

[7] M. T. McCann and M. Unser, "Biomedical image reconstruction: From the foundations to deep neural networks," pp. 283–359, *Foundation and Trends in Signal Processing*, vol. 13, no. 3, 2019. DOI: 10.1561/2000000101.

[8] S. Dempe, "Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints," pp. 333–359, *Optimization*, vol. 52, no. 3, Jun. 2003. DOI: 10.1080/0233193031000149894.

[9] S. Dempe and A. Zemkoho, Eds., *Bilevel Optimization: Advances and next Challenges*, vol. 161, ser. Springer Optimization and Its Applications. Springer International Publishing, 2020. DOI: 10.1007/978-3-030-52119-6.

[10] J. Mairal, F. Bach, and J. Ponce, "Task-driven dictionary learning," pp. 791–804, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, Apr. 2012. DOI: 10.1109/TPAMI.2011.156.

[11] L. Calatroni, C. Chung, J. C. De los Reyes, C.-B. Schönlieb, and T. Valkonen, "Bilevel approaches for learning of variational imaging models," in *Variational Methods in Imaging and Geometric Control*, ser. Radon Series on Computational and Applied Mathematics, vol. 18, De Gruyter, 2017. [Online]. Available: http://arxiv.org/abs/1505.02120.

[12] J. C. De los Reyes, C.-B. Schönlieb, and T. Valkonen, "Bilevel parameter learning for higher-order total variation regularisation models," pp. 1–25, *Journal of Mathematical Imaging and Vision*, vol. 57, no. 1, Jan. 2017. DOI: 10.1007/s10851-016-0662-8.

[13] P. Knöbelreiter, C. Sormann, A. Shekhovtsov, F. Fraundorfer, and T. Pock, "Belief propagation reloaded: Learning BP-layers for labeling problems," presented at the The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7897–7906, Jun. 2020. DOI: 10.1109/CVPR42600.2020.00792.

[14] P. Ochs, R. Ranftl, T. Brox, and T. Pock, "Techniques for gradient-based bilevel optimization with non-smooth lower level problems," pp. 175–194, *Journal of Mathematical Imaging and Vision*, vol. 56, no. 2, Oct. 2016. DOI: 10.1007/s10851-016-0663-7.

[15] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," 65–98, *SIAM Review*, vol. 59, no. 1, 2017. DOI: 10.1137/141000671.

[16] M. Stone, "Cross-validation: A review," 127–139, *Math Oper Stat Ser Stat.*, vol. 9, no. 1, 1978. DOI: 10.1080/02331887808801414.

[17] G. H. Golub, M. Heath, and G. Wahba, "Generalized cross-validation as a method for choosing a good ridge parameter," 215–23, *Technometrics*, vol. 21, no. 2, May 1979. [Online]. Available: http://www.jstor.org/stable/1268518.

[18] D. L. Phillips, "A technique for the numerical solution of certain integral equations of the first kind," 84–97, *J. Assoc. Comput. Mach.*, vol. 9, no. 1, Jan. 1962. DOI: 10.1145/321105.321114.

[19] S. S. Saquib, C. A. Bouman, and K. Sauer, "ML parameter estimation for Markov random fields, with applications to Bayesian tomography," 1029–44, *IEEE Trans. Im. Proc.*, vol. 7, no. 7, Jul. 1998. DOI: 10.1109/83.701163.

[20] W. P. Segars, G. Sturgeon, S. Mendonca, J. Grimes, and B. M. W. Tsui, "4D XCAT phantom for multimodality imaging research," pp. 4902–15, *Medical Physics*, vol. 37, no. 9, Aug. 2010. DOI: 10.1118/1.3480985.

[21] C. Poon and G. Peyré, "Smooth Bilevel Programming for Sparse Regularization," in *35th Conference on Neural Information Processing Systems*, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/0bed45bd5774ffddc95ffe500024f628-Abstract.html.

[22] R. Fletcher and S. Leyffer, "Numerical experience with solving MPECs as NLPs," Department of Mathematics and Computer Science, University of Dundee, Dundee, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.6674.

[23] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," pp. 235–256, *Annals of Operations Research*, vol. 153, no. 1, Jun. 2007. DOI: 10.1007/s10479-007-0176-2.

[24] P. Jain and P. Kar, "Non-convex optimization for machine learning," 142–336, *Found. & Trends in Machine Learning*, vol. 10, no. 3-4, 2017. DOI: 10.1561/2200000058.

[25] A. Effland, E. Kobler, K. Kunisch, and T. Pock, "Variational networks: An optimal control approach to early stopping variational methods for image restoration," pp. 396–416, *Journal of Mathematical Imaging and Vision*, vol. 62, no. 3, Apr. 2020. DOI: 10.1007/s10851-019-00926-8.

[26] E. Haber and L. Tenorio, "Learning regularization functionals a supervised training approach," pp. 611–626, *Inverse Problems*, vol. 19, no. 3, Jun. 1, 2003. DOI: 10.1088/0266-5611/19/3/309.

[27] M. J. Ehrhardt and L. Roberts, "Inexact derivative-free optimization for bilevel learning," pp. 580–600, *Journal of Mathematical Imaging and Vision*, vol. 63, Feb. 6, 2021. DOI: 10.1007/s10851-021-01020-8.

[28] F. Sherry, M. Benning, J. C. De los Reyes, M. J. Graves, G. Maierhofer, G. Williams, C.-B. Schonlieb, and M. J. Ehrhardt, "Learning the sampling pattern for MRI," pp. 4310–4321, *IEEE Transactions on Medical Imaging*, vol. 39, no. 12, Dec. 2020. DOI: 10.1109/TMI.2020.3017353.

[29] E. Kobler, A. Effland, K. Kunisch, and T. Pock, "Total deep variation: A stable regularization method for inverse problems," *IEEE transactions on pattern analysis and machine intelligence*, Nov. 2021. DOI: 10.1109/TPAMI.2021.3124086, Advance online publication. PMID: 34727026.

[30] Y. Chen, R. Ranftl, and T. Pock, "Insights into analysis operator learning: From patch-based sparse models to higher order MRFs," pp. 1060–1072, *IEEE Transactions on Image Processing*, vol. 23, no. 3, Mar. 2014. DOI: 10.1109/TIP.2014.2299065.

[31] K. G. G. Samuel and M. F. Tappen, "Learning optimized MAP estimates in continuously-valued MRF models," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 477–484, Jun. 2009. DOI: 10.1109/CVPR.2009.5206774.

[32] G. Holler, K. Kunisch, and R. C. Barnard, "A bilevel approach for parameter learning in inverse problems," p. 115 012, *Inverse Problems*, vol. 34, no. 11, Nov. 1, 2018. DOI: 10.1088/1361-6420/aade77.

[33] G. Peyré and J. M. Fadili, "Learning analysis sparsity priors," in *IEEE Intl. Conf. on Sampling Theory and Appl. (SampTA)*, 2011. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00542016.

[34] Y. Chen, H. Liu, X. Ye, and Q. Zhang, "Learnable descent algorithm for nonsmooth nonconvex image reconstruction," pp. 1532–1564, *SIAM Journal on Imaging Sciences*, vol. 14, no. 4, 2021. DOI: 10.1137/20M1353368.

[35] R. M. Lewitt and S. Matej, "Overview of methods for image reconstruction from projections in emission computed tomography," 1588–611, *Proc. IEEE*, vol. 91, no. 10, Oct. 2003. DOI: 10.1109/JPROC.2003.817882.

[36] M. Elad, P. Milanfar, and R. Rubinstein, "Analysis versus synthesis in signal priors," pp. 947–68, *Inverse Problems*, vol. 23, no. 3, Jun. 2007. DOI: 10.1088/0266-5611/23/3/007.

[37] C. Guillemot and O. Le Meur, "Image inpainting: Overview and recent advances," 127–44, *IEEE Sig. Proc. Mag.*, vol. 31, no. 1, Jan. 2014. DOI: 10.1109/MSP.2013.2273004.

[38] J. A. Fessler, "Model-based image reconstruction for MRI," 81–9, *IEEE Sig. Proc. Mag.*, vol. 27, no. 4, Jul. 2010. DOI: 10.1109/MSP.2010.936726.

[39] G. H. Golub and C. F. Van Loan, "An analysis of the total least squares problem," 883–93, *SIAM J. Numer. Anal.*, vol. 17, no. 6, Dec. 1980. DOI: 10.1137/0717073.

[40] L. Ying and J. Sheng, "Joint image reconstruction and sensitivity estimation in SENSE (JSENSE)," 1196–1202, *Mag. Res. Med.*, vol. 57, no. 6, Jun. 2007. DOI: 10.1002/mrm.21245.

[41] A. Chambolle and T. Pock, "An introduction to continuous optimization for imaging," pp. 161–319, *Acta Numerica*, vol. 25, May 2016. DOI: 10.1017/S096249291600009X.

[42] S. Nam, M. Davies, M. Elad, and R. Gribonval, "The cosparse analysis model and algorithms," pp. 30–56, *Applied and Computational Harmonic Analysis*, vol. 34, no. 1, Jan. 2013. DOI: 10.1016/j.acha.2012.03.006.

[43] M. Elad, *Sparse and redundant representations: from theory to applications in signal and image processing*. Berlin: Springer, 2010. DOI: 10.1007/978-1-4419-7011-4.

[44] G. Peyre, "A review of adaptive image representations," 896–911, *IEEE J. Sel. Top. Sig. Proc.*, vol. 5, no. 5, Sep. 2011. DOI: 10.1109/JSTSP.2011.2120592.

[45] E. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," pp. 489–509, *IEEE Transactions on Information Theory*, vol. 52, no. 2, Feb. 2006. DOI: 10.1109/TIT.2005.862083.

[46] R. Tibshirani, "Regression shrinkage and selection via the Lasso," pp. 267–288, *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, Jan. 1996. DOI: 10.1111/j.2517-6161.1996.tb02080.x.

[47] P. Zhou, C. Zhang, and Z. Lin, "Bilevel model-based discriminative dictionary learning for recognition," 1173–87, *IEEE Trans. Im. Proc.*, vol. 26, no. 3, Mar. 2017. DOI: 10.1109/tip.2016.2623487.

[48] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithm," 259–68, *Physica D*, vol. 60, no. 1-4, Nov. 1992. DOI: 10.1016/0167-2789(92)90242-F.

[49] E. J. Candès, Y. C. Eldar, D. Needell, and P. Randall, "Compressed sensing with coherent and redundant dictionaries," pp. 59–73, *Applied and Computational Harmonic Analysis*, vol. 31, no. 1, Jul. 2011. DOI: 10.1016/j.acha.2010.10.002.

[50] S. Hawe, M. Kleinsteuber, and K. Diepold, "Analysis operator learning and its application to image reconstruction," pp. 2138–2150, *IEEE Transactions on Image Processing*, vol. 22, no. 6, Jun. 2013. DOI: 10.1109/TIP.2013.2246175.

[51] S. Ravishankar and Y. Bresler, "Learning sparsifying transforms," pp. 1072–1086, *IEEE Transactions on Signal Processing*, vol. 61, no. 5, Mar. 2013. DOI: 10.1109/TSP.2012.2226449.

[52] J. A. Fessler, "Optimization methods for MR image reconstruction," 33–40, *IEEE Sig. Proc. Mag.*, vol. 37, no. 1, Jan. 2020. DOI: 10.1109/MSP.2019.2943645.

[53] L. Pfister and Y. Bresler, "Learning filter bank sparsifying transforms," pp. 504–519, *IEEE Transactions on Signal Processing*, vol. 67, no. 2, Jan. 2019. DOI: 10.1109/TSP.2018.2883021.

[54] B. M. Afkham, J. Chung, and M. Chung, "Learning regularization parameters of inverse problems via deep neural networks," p. 105 017, *Inverse Problems*, vol. 37, no. 10, Sep. 2021. DOI: 10.1088/1361-6420/ac245d.

[55] S. Roth and M. Black, "Fields of experts: A framework for learning image priors," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 860–867, 2005. DOI: 10.1109/CVPR.2005.160.

[56] M. F. Tappen, C. Liu, E. H. Adelson, and W. T. Freeman, "Learning gaussian conditional random fields for low-level vision," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, Jun. 2007. DOI: 10.1109/CVPR.2007.382979.

[57] B. Ophir, M. Elad, N. Bertin, and M. D. Plumbley, "Sequential minimal eigenvalues - an approach to analysis dictionary learning," pp. 1465–1469, *19th European Signal Processing Conference*, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/7074010.

[58] M. Yaghoobi, S. Nam, R. Gribonval, and M. E. Davies, "Analysis operator learning for overcomplete cosparse representations," presented at the 2011 19th European Signal Processing Conference, pp. 1470–1474, IEEE, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/7074220.

[59] M. Yaghoobi, S. Nam, R. Gribonval, and M. E. Davies, "Constrained overcomplete analysis operator learning for cosparse signal modelling," pp. 2341–2355, *IEEE Transactions on Signal Processing*, vol. 61, no. 9, May 2013. DOI: 10.1109/TSP.2013.2250968.

[60] K. Kunisch and T. Pock, "A bilevel optimization approach for parameter learning in variational models," pp. 938–983, *SIAM Journal on Imaging Sciences*, vol. 6, no. 2, Jan. 2013. DOI: 10.1137/120882706.

[61] I. Y. Chun and J. A. Fessler, "Convolutional analysis operator learning: Acceleration and convergence," pp. 2108–2122, *IEEE Transactions on Image Processing*, vol. 29, 2020. DOI: 10.1109/TIP.2019.2937734.

[62] S. Haykin, "Neural networks expand SP's horizons," 24–49, *IEEE Sig. Proc. Mag.*, vol. 13, no. 2, Mar. 1996. DOI: 10.1109/79.487040.

[63] J.-N. Hwang, S.-Y. Kung, M. Niranjan, and J. C. Principe, "The past, present, and future of neural networks for signal processing," 28–48, *IEEE Sig. Proc. Mag.*, vol. 14, no. 6, Nov. 1997. DOI: 10.1109/79.637299.

[64] A. Lucas, M. Iliadis, R. Molina, and A. K. Katsaggelos, "Using deep neural networks for inverse problems in imaging: Beyond analytical methods," 20–36, *IEEE Sig. Proc. Mag.*, vol. 35, no. 1, Jan. 2018. DOI: 10.1109/msp.2017.2760358.

[65] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention*, 234–41, 2015. DOI: 10.1007/978-3-319-24574-4_28.

[66] J. C. Ye, Y. Han, and E. Cha, "Deep convolutional framelets: A general deep learning framework for inverse problems," 991–1048, *SIAM J. Imaging Sci.*, vol. 11, no. 2, Jan. 2018. DOI: 10.1137/17m1141771.

[67] B. Wen, S. Ravishankar, L. Pfister, and Y. Bresler, "Transform learning for magnetic resonance image reconstruction: From model-based learning to building neural networks," 41–53, *IEEE Sig. Proc. Mag.*, vol. 37, no. 1, Jan. 2020. DOI: 10.1109/MSP.2019.2951469.

[68] A. Chambolle and T. Pock, "Learning consistent discretizations of the total variation," pp. 778–813, vol. 14, no. 2, 2021. DOI: 10.1137/20M1377199.

[69] M. Feurer and F. Hutter, "Chapter 1: Hyperparameter optimization," in *Automated Machine Learning: Methods, Systems, Challenges*, ser. The Springer Series on Challenges in Machine Learning, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., Springer International Publishing, 2019, pp. 3–33. DOI: 10.1007/978-3-030-05318-5.

[70] L. A. Shepp and B. F. Logan, "The Fourier reconstruction of a head section," 21–43, *IEEE Trans. Nuc. Sci.*, vol. 21, no. 3, Jun. 1974. DOI: 10.1109/TNS.1974.6499235.

[71] J. A. Fessler, *MIRT-demo: 01-recon*, Jul. 25, 2020. [Online]. Available: https://github.com/JeffFessler/mirt-demo/blob/master/isbi-19/01-recon.jl.

[72] C. You, Q. Yang, H. Shan, L. Gjesteby, G. Li, S. Ju, Z. Zhang, Z. Zhao, Y. Zhang, W. Cong, and G. Wang, "Structure-sensitive multi-scale deep neural network for low-dose CT denoising," pp. 41 839–41 855, *IEEE Access*, vol. 6, 2018. DOI: 10.1109/ACCESS.2018.2858196.

[73] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," 600–12, *IEEE Trans. Im. Proc.*, vol. 13, no. 4, Apr. 2004. DOI: 10.1109/TIP.2003.819861.

[74] Z. Wang and A. Bovik, "Reduced- and no-reference image quality assessment," pp. 29–40, *IEEE Signal Processing Magazine*, vol. 28, no. 6, Nov. 2011. DOI: 10.1109/MSP.2011.942471.

[75] L. Zhang, L. Zhang, X. Mou, and D. Zhang, "A comprehensive evaluation of full reference image quality assessment algorithms," in *2012 19th IEEE International Conference on Image Processing*, pp. 1477–1480, Sep. 2012. DOI: 10.1109/ICIP.2012.6467150.

[76] W. Zhang, K. Ma, J. Yan, D. Deng, and Z. Wang, "Blind image quality assessment using a deep bilinear convolutional neural network," pp. 36–47, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 1, Jan. 2020. DOI: 10.1109/TCSVT.2018.2886771.

[77] A. Mason, J. Rioux, S. E. Clarke, A. Costa, M. Schmidt, V. Keough, T. Huynh, and S. Beyea, "Comparison of objective image quality metrics to expert radiologists' scoring of diagnostic quality of MR images," pp. 1064–1072, *IEEE Transactions on Medical Imaging*, vol. 39, no. 4, Apr. 2020. DOI: 10.1109/TMI.2019.2930338.

[78] M. Gholizadeh-Ansari, J. Alirezaie, and P. Babyn, "Deep learning for low-dose CT denoising using perceptual loss and edge detection layer," 504–15, *J. Digital Im.*, vol. 33, no. 2, 2020. DOI: 10.1007/s10278-019-00274-4.

[79] G. Seif and D. A., "Edge-based loss function for single image super-resolution," in *Proc. IEEE Conf. Acoust. Speech Sig. Proc.*, 1468–72, 2018. DOI: 10.1109/ICASSP.2018.8461664.

[80] S. Ravishankar and Y. Bresler, "MR image reconstruction from highly undersampled k-space data by dictionary learning," pp. 1028–1041, *IEEE Transactions on Medical Imaging*, vol. 30, no. 5, May 2011. DOI: 10.1109/TMI.2010.2090538.

[81] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," pp. 600–612, *IEEE Transactions on Image Processing*, vol. 13, no. 4, Apr. 2004. DOI: 10.1109/TIP.2003.819861.

[82] Z. Wang, E. Simoncelli, and A. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, pp. 1398–1402, IEEE, 2003. DOI: 10.1109/ACSSC.2003.1292216.

[83] G. P. Renieblas, A. T. Nogués, A. M. González, N. G. León, and E. G. . Castillo, "Structural similarity index family for image quality assessment in radiological images," p. 035 501, *J. Med. Im.*, vol. 4, no. 3, Jul. 2017. DOI: 10.1117/1.JMI.4.3.035501.

[84] S. Bosse, D. Maniry, K.-R. Muller, T. Wiegand, and W. Samek, "Deep neural networks for no-reference and full-reference image quality assessment," pp. 206–219, *IEEE Transactions on Image Processing*, vol. 27, no. 1, Jan. 2018. DOI: 10.1109/TIP.2017.2760518.

[85] G. W. Lindsay, "Convolutional neural networks as a model of the visual system: Past, present, and future," pp. 1–15, *Journal of Cognitive Neuroscience*, Feb. 6, 2020. DOI: 10.1162/jocn_a_01544.

[86] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, May 2015. [Online]. Available: http://arxiv.org/abs/1409.1556.

[87] T. J. Hebert and R. Leahy, "Statistic-based MAP image reconstruction from Poisson data using Gibbs priors," 2290–303, *IEEE Trans. Sig. Proc.*, vol. 40, no. 9, Sep. 1992. DOI: 10.1109/78.157228.

[88] C. M. Stein, "Estimation of the mean of a multivariate normal distribution," *The Annals of Statistics*, vol. 9, no. 6, Nov. 1, 1981. DOI: 10.1214/aos/1176345632.

[89] S. Ramani, T. Blu, and M. Unser, "Monte-carlo sure: A black-box optimization of regularization parameters for general denoising algorithms," pp. 1540–1554, *IEEE Transactions on Image Processing*, vol. 17, no. 9, Sep. 2008. DOI: 10.1109/TIP.2008.2001404.

[90] S. Soltanayev and S. Y. Chun, "Training deep learning based denoisers without ground truth data," in *Neural Information Processing Systems*, vol. 31, 2018. [Online]. Available: https://papers.nips.cc/paper/7587-training-deep-learning-based-denoisers-without-ground-truth-data.

[91] K. Kim, S. Soltanayev, and S. Y. Chun, "Unsupervised training of denoisers for low-dose CT reconstruction without full-dose ground truth," 1112–25, *IEEE J. Sel. Top. Sig. Proc.*, vol. 14, no. 6, Oct. 2020. DOI: 10.1109/JSTSP.2020.3007326.

[92] M. Zhussip, S. Soltanayev, and S. Y. Chun, "Training deep learning based image denoisers from undersampled measurements without ground truth and without image prior," in *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, 10247–56, 2019. DOI: 10.1109/CVPR.2019.01050.

[93] H. Zhang, X. Chen, X. Zhang, and X. Zhang, "A bi-level nested sparse optimization for adaptive mechanical fault feature detection," pp. 19 767–19 782, *IEEE Access*, vol. 8, 2020. DOI: 10.1109/ACCESS.2020.2968726.

[94] C.-A. Deledalle, S. Vaiter, J. Fadili, and G. Peyré, "Stein Unbiased GrAdient estimator of the Risk (SUGAR) for multiple parameter selection," pp. 2448–2487, *SIAM Journal on Imaging Sciences*, vol. 7, no. 4, Jan. 2014. DOI: 10.1137/140968045.

[95] Y. C. Eldar, "Rethinking biased estimation: Improving maximum likelihood and the Cramer-Rao bound," 305–449, *Found. & Trends in Sig. Pro.*, vol. 1, no. 4, 2008. DOI: 10.1561/2000000008.

[96] Y. Eldar, "Generalized SURE for exponential families: Applications to regularization," pp. 471–481, *IEEE Transactions on Signal Processing*, vol. 57, no. 2, Feb. 2009. DOI: 10.1109/TSP.2008.2008212.

[97] R. Giryes, M. Elad, and Y. C. Eldar, "The projected GSURE for automatic parameter tuning in iterative shrinkage methods," 407–22, *Applied and Computational Harmonic Analysis*, vol. 30, no. 3, May 2011. DOI: 10.1016/j.acha.2010.11.005.

[98] A. Mittal, R. Soundararajan, and A. C. Bovik, "Making a "completely blind" image quality analyzer," pp. 209–212, *IEEE Signal Processing Letters*, vol. 20, no. 3, Mar. 2013. DOI: 10.1109/LSP.2012.2227726.

[99] L. Kang, P. Ye, Y. Li, and D. Doermann, "Convolutional neural networks for no-reference image quality assessment," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1733–1740, Jun. 2014. DOI: 10.1109/CVPR.2014.224.

[100] The University of Texas at Austin: Laboratory for Image and Video Engineering. (n.d.). "Image & video quality assessment at LIVE," [Online]. Available: http://live.ece.utexas.edu/research/quality/.

[101] J. Larson, M. Menickelly, and S. M. Wild, "Derivative-free optimization methods," pp. 287–404, *Acta Numerica*, vol. 28, May 1, 2019. DOI: 10.1017/S0962492919000060.

[102] O. Gencoglu, M. van Gils, E. Guldogan, C. Morikawa, M. Süzen, M. Gruber, J. Leinonen, and H. Huttunen. (Apr. 16, 2019). "HARK side of deep learning – From grad student descent to automated machine learning." arXiv: 1904.07633.

[103] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," pp. 281–305, *Journal of Machine Learning Research*, vol. 13, Feb. 2012. DOI: 10.5555/2188385.2188395.

[104] G. Muniraju, B. Kailkhura, J. J. Thiagarajan, and T. Bremer, "Controlled random search improves sample mining and hyper-parameter optimization," in *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019. [Online]. Available: https://www.osti.gov/servlets/purl/1497973.

[105] H.-G. Beyer, *The Theory of Evolution Strategies*, ser. Natural Computing Series. Springer, 2001.

[106] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian hyperparameter optimization on large datasets," pp. 4945–68, *Electron. J. Statist.*, vol. 11, no. 2, 2017. DOI: 10.1214/17-EJS1335SI.

[107] P. I. Frazier. (Jul. 8, 2018). "A tutorial on bayesian optimization." arXiv: 1807.02811.

[108] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust Region Methods*, ser. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Jan. 1, 2000, 960 pp. DOI: 10.1137/1.9780898719857.

[109] L. Roberts, *Inexact DFO for Bilevel Learning: Dimension Question*, E-mail, Jul. 11, 2021.

[110] C. Cartis and L. Roberts. (Feb. 23, 2021). "Scalable subspace methods for derivative-free nonlinear least-squares optimization." arXiv: 2102.12016.

[111] S. Gould, B. Fernando, A. Cherian, P. Anderson, R. S. Cruz, and E. Guo. (Jul. 20, 2016). "On differentiating parameterized argmin and argmax problems with application to bi-level optimization." arXiv: 1607.05447.

[112] J. A. Fessler, "Mean and variance of implicitly defined biased estimators (such as penalized maximum likelihood): Applications to tomography," pp. 493–506, *IEEE Trans. Im. Proc.*, vol. 5, no. 3, Mar. 1996. DOI: 10.1109/83.491322.

[113] M. Hintermüller and T. Wu, "Bilevel optimization for calibrating point spread functions in blind deconvolution," pp. 1139–1169, *Inverse Problems & Imaging*, vol. 9, no. 4, 2015. DOI: 10.3934/ipi.2015.9.1139.

[114] S. Scholtes and M. Stöhr, "How stringent is the linear independence assumption for mathematical programs with complementarity constraints?" Pp. 851–863, *Mathematics of Operations Research*, vol. 26, no. 4, Nov. 2001. DOI: 10.1287/moor.26.4.851.10007.

[115] S. Dempe and J. Dutta, "Is bilevel programming a special case of a mathematical program with complementarity constraints?" Pp. 37–48, *Mathematical Programming*, vol. 131, no. 1-2, Feb. 2012. DOI: 10.1007/s10107-010-0342-1.

[116] K. B. Petersen and M. S. Pedersen, *The Matrix Cookbook*. Technical University of Denmark, Nov. 2012. [Online]. Available: http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=3274.

[117] K. Ji, J. Yang, and Y. Liang, "Bilevel optimization: Convergence analysis and enhanced design," in *Proceedings of the 38th International Conference on Machine Learning*, pp. 4882–4892, Jul. 2021. [Online]. Available: http://proceedings.mlr.press/v139/ji21c.html.

[118] R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo, "On the iteration complexity of hypergradient computation," in *Proceedings of the 37th International Conference on Machine Learning*, p. 11, 2020. [Online]. Available: http://proceedings.mlr.press/v119/grazzi20a.html.

[119] P. W. Holland and R. E. Welsch, "Robust regression using iteratively reweighted least-squares," 813–27, *Comm. in Statistics—Theory and Methods*, vol. 6, no. 9, 1977. DOI: 10.1080/03610927708827533.

[120] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," 301–20, *J. Royal Stat. Soc. Ser. B*, vol. 67, no. 2, 2005. DOI: 10.1111/j.1467-9868.2005.00503.x.

[121] C.-s. Foo, C. B., and A. Ng, "Efficient multiple hyperparameter learning for log-linear models," in *Advances in Neural Information Processing Systems*, vol. 20, Curran Associates, Inc., 2007. [Online]. Available: https://proceedings.neurips.cc/paper/2007/hash/851ddf5058cf22df63d3344ad89919cf-Abstract.html.

[122] Z. Ramzi, F. Mannel, S. Bai, J.-L. Starck, P. Ciuciu, and T. Moreau, *SHINE: SHaring the INverse Estimate from the forward pass for bi-level optimization and implicit models*, Jun. 24, 2021. arXiv: 2106.00553.

[123] P. Sprechmann, R. Litman, T. B. Yakar, A. M. Bronstein, and G. Sapiro, "Supervised sparse analysis and synthesis operators," in *Neural Information Processing Systems*, pp. 908–916, 2013. [Online]. Available: https://papers.nips.cc/paper/2013/hash/7380ad8a673226ae47fce7bff88e9c33-Abstract.html.

[124] M. T. McCann and S. Ravishankar, "Supervised learning of sparsity-promoting regularizers for denoising," *arXiv Computing Research Repository*, Jun. 9, 2020. arXiv: 2006.05521.

[125] A. Ghosh, M. T. Mccann, and S. Ravishankar, *Bilevel learning of l1-regularizers with closed-form gradients(BLORC)*, Nov. 21, 2021. arXiv: 2111.10858.

[126] R. J. Tibshirani and J. Taylor, "The solution path of the generalized lasso," *The Annals of Statistics*, vol. 39, no. 3, Jun. 2011. DOI: 10.1214/11-AOS878.

[127] A. Ghosh, *Questions about BLORC*, E-mail, Feb. 21, 2022.

[128] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *Proceedings of the International Conference on Machine Learning*, pp. 1165–1173, PMLR, Dec. 12, 2017. [Online]. Available: http://proceedings.mlr.press/v70/franceschi17a.html.

[129] B. Dauvergne and L. Hascoet, "The data-flow equations of checkpointing in reverse automatic differentiation," in *International Conference on Computational Science*, pp. 566–573, 2006. DOI: 10.1007/11758549_78.

[130] M. Kellman, K. Zhang, E. Markley, J. Tamir, E. Bostan, M. Lustig, and L. Waller, "Memory-efficient learning for large-scale computational imaging," pp. 1403–1414, *IEEE Transactions on Computational Imaging*, vol. 6, 2020. DOI: 10.1109/TCI.2020.3025735.

[131] D. Gilton, G. Ongie, and R. Willett, "Model adaptation for inverse problems in imaging," pp. 661–674, *IEEE Transactions on Computational Imaging*, vol. 7, 2021. DOI: 10.1109/TCI.2021.3094714.

[132] H. Antil, Z. Di, and R. Khatri, "Bilevel optimization, deep learning and fractional laplacian regularization with applications in tomography," *Inverse Problems*, Mar. 18, 2020. DOI: 10.1088/1361-6420/ab80d7.

[133] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural Ordinary Differential Equations," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018. [Online]. Available: https://papers.nips.cc/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html.

[134] M. Thies, F. Wagner, M. Gu, L. Folle, L. Felsner, and A. Maier, *Learned Cone-Beam CT Reconstruction Using Neural Ordinary Differential Equations*, Jan. 19, 2022. arXiv: 2201.07562.

[135] A. Chambolle and T. Pock, "On the ergodic convergence rates of a first-order primal—dual algorithm," pp. 253–287, *Mathematical Programming: Series A and B*, vol. 159, no. 1-2, Sep. 2016. DOI: 10.1007/s10107-015-0957-3.

[136] C. Christof, "Gradient-based solution algorithms for a class of bilevel optimization and optimal control problems with a nonsmooth lower level," pp. 290–318, *SIAM Journal on Optimization*, vol. 30, no. 1, Jan. 2020. DOI: 10.1137/18M1225707.

[137] A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots, "Truncated back-propagation for bilevel optimization," in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, pp. 1723–1732, PMLR, Apr. 11, 2019. [Online]. Available: https://proceedings.mlr.press/v89/shaban19a.html.

[138] D. P. Palomar and Y. C. Eldar, *Convex optimization in signal processing and communications*. Cambridge, 2011. DOI: 10.1017/CBO9780511804458.

[139] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in *Proceedings International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48, pp. 737–46, PMLR, Jun. 20–22, 2016. [Online]. Available: http://proceedings.mlr.press/v48/pedregosa16.html.

[140] Y. Chen, T. Pock, R. Ranftl, and H. Bischof, "Revisiting loss-specific training of filter-based MRFs for image restoration," in *Pattern Recognition*, J. Weickert, M. Hein, and B. Schiele, Eds., pp. 271–281, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-40602-7_30.

[141] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *International Conference on Machine Learning*, pp. 1568–1577, PMLR, Jul. 3, 2018. [Online]. Available: http://proceedings.mlr.press/v80/franceschi18a.html.

[142] M. Hintermüller, K. Papafitsoros, C. N. Rautenberg, and H. Sun. (Feb. 13, 2020). "Dualization and automatic distributed parameter selection of total generalized variation via bilevel optimization." arXiv: 2002.05614.

[143] B. Sixou, "Adaptative regularization parameter for poisson noise with a bilevel approach: Application to spectral computerized tomography," pp. 1–18, *Inverse Problems in Science and Engineering*, Dec. 22, 2020. DOI: 10.1080/17415977.2020.1864348.

[144] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," 1190–208, *SIAM J. Sci. Comp.*, vol. 16, no. 5, 1995. DOI: 10.1137/0916069.

[145] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations*, vol. abs/1412.6980, May 2015. arXiv: 1412.6980.

[146] J. Fehrenbach, M. Nikolova, G. Steidl, and P. Weiss, "Bilevel image denoising using gaussianity tests," in *International Conference on Scale Space and Variational Methods in Computer Vision*, vol. 9087, pp. 117–128, 2015. DOI: 10.1007/978-3-319-18461-6_10.

[147] B. Lecouat, J. Ponce, and J. Mairal, "A flexible framework for designing trainable priors with adaptive smoothing and game encoding," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 664–15 675, 2020. [Online]. Available: https://papers.nips.cc/paper/2020/hash/b4edda67f0f57e218a8e766927e3e5c5-Abstract.html.

[148] D. Kim and J. A. Fessler, "Adaptive restart of the optimized gradient method for convex optimization," 240–63, *J. Optim. Theory Appl.*, vol. 178, no. 1, Jul. 2018. DOI: 10.1007/s10957-018-1287-4.

[149] M. Hong, H.-T. Wai, Z. Wang, and Z. Yang, *A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic*, Dec. 20, 2020. arXiv: 2007.05170.

[150] T. Chen, Y. Sun, and W. Yin. (Feb. 22, 2021). "A single-timescale stochastic bilevel optimization method." arXiv: 2102.04671.

[151] S. Ghadimi and M. Wang. (Feb. 6, 2018). "Approximation methods for bilevel programming." arXiv: 1802.02246.

[152] J. Yang, K. Ji, and Y. Liang, "Provably faster algorithms for bilevel optimization," in *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/71cc107d2e0408e60a3d3c44f47507bd-Abstract.html.

[153] P. Khanduri, H.-T. Wai, S. Zeng, M. Hong, Z. Wang, and Z. Yang, "A near-optimal algorithm for stochastic bilevel optimization via double-momentum," in *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, p. 13, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/fe2b421b8b5f0e7c355ace66a9fe0206-Abstract.html.

[154] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$," 372–76, *Soviet Math. Dokl.*, vol. 27, no. 2, 1983.

[155] L. M. Nguyen, J. Liu, K. Scheinberg, and M. Takác, "SARAH: A novel method for machine learning problems using stochastic recursive gradient," in *34th International Conference on Machine Learning*, p. 9, 2017. [Online]. Available: https://proceedings.mlr.press/v70/nguyen17b.html.

[156] A. Mehra and J. Hamm, "Penalty method for inversion-free deep bilevel optimization," in *Proceedings of The 13th Asian Conference on Machine Learning*, pp. 347–362, PMLR, Nov. 28, 2021. [Online]. Available: https://proceedings.mlr.press/v157/mehra21a.html.

[157] L. Hoeltgen, S. Setzer, and J. Weickert, "An optimal control approach to find sparse data for Laplace interpolation," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, A. Heyden, F. Kahl, C. Olsson, M. Oskarsson, and X.-C. Tai, Eds., vol. 8081, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 151–164. DOI: 10.1007/978-3-642-40395-8_12.

[158] D. Kim and J. A. Fessler, "On the convergence analysis of the optimized gradient method," pp. 187–205, *Journal of Optimization Theory and Applications*, vol. 172, no. 1, Jan. 2017. DOI: 10.1007/s10957-016-1018-7.

[159] Y. Drori, "The exact information-based complexity of smooth convex minimization," 1–16, *J. Complexity*, vol. 39, Apr. 2017. DOI: 10.1016/j.jco.2016.11.001.

[160] S. Nowozin, "Structured learning and prediction in computer vision," pp. 185–365, *Foundations and Trends® in Computer Graphics and Vision*, vol. 6, no. 3-4, 2011. DOI: 10.1561/0600000033.

[161] M. Nikolova and ,CMLA, ENS Cachan, CNRS, PRES UniverSud, 61 Av. President Wilson, F-94230 Cachan, "Model distortions in Bayesian MAP reconstruction," pp. 399–422, *Inverse Problems & Imaging*, vol. 1, no. 2, 2007. DOI: 10.3934/ipi.2007.1.399.

[162] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-D transform-domain collaborative filtering," pp. 2080–2095, *IEEE Transactions on Image Processing*, vol. 16, no. 8, Aug. 2007. DOI: 10.1109/TIP.2007.901238.

[163] K. Bredies, K. Kunisch, and T. Pock, "Total generalized variation," pp. 492–526, *SIAM Journal on Imaging Sciences*, vol. 3, no. 3, Jan. 2010. DOI: 10.1137/090769521.

[164] A. Chambolle and P.-L. Lions, "Image recovery via total variation minimization and related problems," pp. 167–188, *Numerische Mathematik*, vol. 76, no. 2, Apr. 1, 1997. DOI: 10.1007/s002110050258.

[165] M. Benning, C. Brune, M. Burger, and J. Müller, "Higher-order TV methods—Enhancement via Bregman iteration," pp. 269–310, *Journal of Scientific Computing*, vol. 54, no. 2-3, Feb. 2013. DOI: 10.1007/s10915-012-9650-3.

[166] F. Knoll, K. Bredies, T. Pock, and R. Stollberger, "Second order total generalized variation (TGV) for MRI," 480–91, *Mag. Res. Med.*, vol. 65, no. 2, 2011. DOI: 10.1002/mrm.22595.

[167] S. Setzer, G. Steidl, and T. Teuber, "Infimal convolution regularizations with discrete $\ell 1$-type functionals," 797–827, *Comm. Math. Sci.*, vol. 9, no. 3, 2011. DOI: 10.4310/CMS.2011.v9.n3.a7.

[168] M. D'Elia, J. C. De los Reyes, and A. M. Trujillo, *Bilevel parameter optimization for learning nonlocal image denoising models*, Apr. 29, 2020. arXiv: 1912.02347.

[169] B. Gozcu, R. K. Mahabadi, Y.-H. Li, E. Ilicak, T. Cukur, J. Scarlett, and V. Cevher, "Learning-based compressive MRI," 1394–406, *IEEE Trans. Med. Imag.*, vol. 37, no. 6, Jun. 2018. DOI: 10.1109/TMI.2018.2832540.

[170] N. Shlezinger, J. Whang, Y. C. Eldar, and A. G. Dimakis, *Model-based deep learning*, Dec. 15, 2020. arXiv: 2012.08405.

[171] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," pp. 18–44, *IEEE Signal Processing Magazine*, vol. 38, no. 2, Mar. 2021. DOI: 10.1109/MSP.2020.3016905.

[172] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. Intl. Conf. Mach. Learn*, 2010. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/gregor-icml-10.pdf.

[173] W. Bian, Y. Chen, and X. Ye, "Deep parallel MRI reconstruction network without coil sensitivities," in *Machine Learning for Medical Image Reconstruction*, F. Deeba, P. Johnson, T. Würfl, and J. C. Ye, Eds., ser. Lecture Notes in Computer Science, pp. 17–26, Springer International Publishing, 2020. DOI: 10.1007/978-3-030-61598-7_2.

[174] K. Hammernik, T. Klatzer, E. Kobler, M. P. Recht, D. K. Sodickson, T. Pock, and F. Knoll, "Learning a variational network for reconstruction of accelerated MRI data," pp. 3055–3071, *Magnetic Resonance in Medicine*, vol. 79, no. 6, 2018. DOI: 10.1002/mrm.26977.

[175] H. Lim, I. Y. Chun, Y. K. Dewaraja, and J. A. Fessler, "Improved low-count quantitative PET reconstruction with an iterative neural network," pp. 3512–3522, *IEEE Transactions on Medical Imaging*, vol. 39, no. 11, Nov. 2020. DOI: 10.1109/TMI.2020.2998480.

[176] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/01386bd6d8e091c2ab4c7c7de644d37b-Abstract.html.

[177] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552, PMLR, Jun. 3, 2020. [Online]. Available: https://proceedings.mlr.press/v108/lorraine20a.html.

[178] S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin, "JFB: Jacobian-free backpropagation for implicit networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022. arXiv: 2103.12803.

[179] H. Heaton, S. Wu Fung, A. Gibali, and W. Yin, "Feasibility-based fixed point networks," p. 21, *Fixed Point Theory and Algorithms for Sciences and Engineering*, vol. 2021, no. 1, Dec. 2021. DOI: 10.1186/s13663-021-00706-3.

[180] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg, "Plug-and-Play priors for model based reconstruction," in *2013 IEEE Global Conference on Signal and Information Processing*, pp. 945–948, IEEE, Dec. 2013. DOI: 10.1109/GlobalSIP.2013.6737048.

[181] J. Eckstein and D. P. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," 293–318, *Mathematical Programming*, vol. 55, no. 1-3, Apr. 1992. DOI: 10.1007/BF01581204.

[182] J. He, Y. Yang, Y. Wang, D. Zeng, Z. Bian, H. Zhang, J. Sun, Z. Xu, and J. Ma, "Optimizing a parameterized plug-and-play ADMM for iterative low-dose CT reconstruction," pp. 371–382, *IEEE Transactions on Medical Imaging*, vol. 38, no. 2, Feb. 2019. DOI: 10.1109/TMI.2018.2865202.

[183] C. Crockett, D. Hong, I. Y. Chun, and J. A. Fessler, "Incorporating handcrafted filters in convolutional analysis operator learning for ill-posed inverse problems," in *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pp. 316–320, Dec. 2019. DOI: 10.1109/CAMSAP45676.2019.9022669.

[184] H. H. Barrett, "Objective assessment of image quality: Effects of quantum noise and object variability," 1266–1278, *J. Opt. Soc. Am. A*, vol. 7, no. 7, Jul. 1990. DOI: 10.1364/JOSAA.7.001266.

[185] A. Yendiki and J. A. Fessler, "Analysis of observer performance in unknown-location tasks for tomographic image reconstruction," B99–109, *J. Opt. Soc. Am. A*, vol. 24, no. 12, Dec. 2007. DOI: 10.1364/JOSAA.24.000B99.

[186] F. K. Kopp, M. Catalano, D. Pfeiffer, A. A. Fingerle, E. J. Rummeny, and P. B. Noel, "CNN as model observer in a liver lesion detection task for x-ray computed tomography: A phantom study," 4439–47, *Med. Phys.*, vol. 45, no. 10, Oct. 2018. DOI: 10.1002/mp.13151.

[187] J. Xu and F. Noo, "Patient-specific hyperparameter learning for optimization-based CT image reconstruction," 19NT01, *Physics in Medicine & Biology*, vol. 66, no. 19, Sep. 2021. DOI: 10.1088/1361-6560/ac0f9a.

[188] J. Mairal, G. Sapiro, and M. Elad, "Learning multiscale sparse representations for image and video restoration," pp. 214–241, *Multiscale Modeling & Simulation*, vol. 7, no. 1, Jan. 2008. DOI: 10.1137/070697653.

[189] T. Liu, A. Chaman, D. Belius, and I. Dokmanić, *Learning multiscale convolutional dictionaries for image reconstruction*, Aug. 19, 2021. arXiv: 2011.12815.

[190] C. Crockett and J. A. Fessler, "Motivating bilevel approaches to filter learning: A case study," in *2021 IEEE International Conference on Image Processing (ICIP)*, pp. 2803–2807, IEEE, Sep. 19, 2021. DOI: 10.1109/ICIP42928.2021.9506489.

[191] J. Kaipioa and E. Somersalo, "Statistical inverse problems: Discretization, model reduction and inverse crimes," 493–504, *J. Comp. Appl. Math.*, vol. 198, no. 2, Jan. 2007. DOI: 10.1016/j.cam.2005.09.027.

[192] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, "Deep convolutional neural network for inverse problems in imaging," 4509–22, *IEEE Trans. Im. Proc.*, vol. 26, no. 9, Sep. 2017. DOI: 10.1109/TIP.2017.2713099.

[193] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," p. 60, *J. Big Data*, vol. 6, no. 1, Jul. 2019. DOI: 10.1186/s40537-019-0197-0.

[194] J. A. Fessler and W. L. Rogers, "Spatial resolution properties of penalized-likelihood image reconstruction methods: Space-invariant tomographs," 1346–58, *IEEE Trans. Im. Proc.*, vol. 5, no. 9, Sep. 1996. DOI: 10.1109/83.535846.

[195] J. Qi and R. H. Huesman, "Penalized maximum-likelihood image reconstruction for lesion detection," 4017–30, *Phys. Med. Biol.*, vol. 51, no. 16, Aug. 2006. DOI: 10.1088/0031-9155/51/16/009.

[196] L. Yang, J. Zhou, A. Ferrero, R. D. Badawi, and J. Qi, "Regularization design in penalized maximum-likelihood image reconstruction for lesion detection in 3D PET," 403–20, *Phys. Med. Biol.*, vol. 59, no. 2, Jan. 2014. DOI: 10.1088/0031-9155/59/2/403.

[197] B. Sahiner, A. Pezeshk, L. M. Hadjiiski, X. Wang, K. Drukker, K. Cha, R. Summers, and M. L. Giger, "Deep learning in medical imaging and radiation therapy," *Medical Physics*, Nov. 2018. DOI: 10.1002/mp.13264.

[198] FDA, *510k premarket notification of Deep Learning Image Reconstruction (GE Medical Systems)*, 2019. [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfpmn/pmn.cfm?ID=K183202.

[199] J. Solomon, P. Lyu, D. Marin, and E. Samei, "Noise and spatial resolution properties of a commercially available deep learning-based CT reconstruction algorithm," 3961–71, *Med. Phys.*, vol. 47, no. 9, 2020. DOI: 10.1002/mp.14319.

[200] C. Garcia-Cardona and B. Wohlberg, "Convolutional dictionary dearning: A comparative review and new algorithms," pp. 366–381, *IEEE Transactions on Computational Imaging*, vol. 4, no. 3, Sep. 2018. DOI: 10.1109/TCI.2018.2840334.

[201] I. Y. Chun and J. A. Fessler, "Convolutional dictionary learning: Acceleration and convergence," 1697–712, *IEEE Trans. Im. Proc.*, vol. 27, no. 4, Apr. 2018. DOI: 10.1109/TIP.2017.2761545.

[202] "Fenchel Duality," in *Convex Analysis and Nonlinear Optimization: Theoryand Examples*, ser. CMS Books in Mathematics, J. Borwein and A. Lewis, Eds., New York, NY: Springer, 2006, pp. 33–63. DOI: 10.1007/978-0-387-31256-9_3.

[203] M. Unser and T. Blu, "Generalized smoothing splines and the optimal discretization of the Wiener filter," 2146–59, *IEEE Trans. Sig. Proc.*, vol. 53, no. 6, Jun. 2005. DOI: 10.1109/TSP.2005.847821.

[204] C. Crockett, *BilevelFilterLearningForImageRecon*, 2022. [Online]. Available: https://github.com/cecroc/BilevelFilterLearningForImageRecon.